Department of Computer Science

Technical Report

Undergraduate Dissertation: Creating an extensible Unit Converter using OpenMath as the Representation of the Semantics of the Units

Jonathan Stratford

Technical Report 2008-02 **June 2008**

ISSN 1740-9497

Contact Address:
Department of Computer Science
University of Bath
Bath, BA2 7AY
United Kingdom
URL: http://www.cs.bath.ac.uk

ISSN 1740-9497

Creating an extensible Unit Converter using OpenMath as the Representation of the Semantics of the Units

Jonathan Stratford

Bachelor of Science in Computer Science with Honours
The University of Bath
April 2008

Creating an extensible Unit Converter using OpenMath as the Representation of the Semantics of the Units

Submitted by: Jonathan Stratford

COPYRIGHT

Declaration

This dissertation is submitted to the University of Bath in accordance with the requirements of the degree of Bachelor of Science in the Department of Computer Science. No portion of the work in this dissertation has been submitted in support of an application for any other degree or qualification of this or any other university or institution of learning. Except where specifically acknowledged, it is the work of the author.

Signed:

Abstract

Present web-based unit converters lack several useful features. This project investigates the use of OpenMath in this area and produces a working web-based unit converter that allows a user to add new units on a temporary basis. Functionality is also provided to convert to a particular measurement standard, where the system chooses the most appropriate units to use based on the input unit and quantity. Various deficiencies in OpenMath's unit support are discussed, and possibilities for improvement are considered.

Contents

List of Figures

List of Tables

Acknowledgements

I am very grateful to Professor James Davenport for supporting me throughout this project. Additionally, my parents, Mike and Jill Stratford have been very supportive throughout, and have always been happy to proofread my work.

Chapter 1

Introduction

Unit conversion in general is largely a solved problem: many web-based unit converters are publicly available, which can convert between units perfectly adequately. Why, then, is this project investigating the area? A common limitation of all existing web-based systems is that the user is unable to add new units: they are limited to the set of units provided by the author, and this set, however large, will not contain *every* possible unit that a user may want, and in some cases may have far too many that the user *does not* want. This is where, we believe, this project can fill the void. The system proposed for this project uses OpenMath 2.0 along with its Simple Type System (STS) to store the semantics of units in so-called Content Dictionaries (CDs), to be used in conversions. The main philosophy of OpenMath (Buswell, Caprotti, Carlisle, Dewar, Gaëtano & Kohlhase 2004) is transmitting mathematical semantics, between programs or across the web, for example. For this process an XML format is used, and therefore, the proposed system could accept new unit definitions from users in the form of uploaded XML CD files. This makes the system infinitely extensible: any unit that can be described in a Content Dictionary can be given to the system, or the OpenMath website, which holds the central repository. If a user thinks that their new CD would be helpful to other users, they can submit it to the OpenMath community so that, if/once approved, anyone can use it. If the new CD is for a very specialist application, they could choose not to submit it, but just provide it to the proposed system on a temporary basis.

Other features that OpenMath with STS provides include the ability to convert to an appropriate set of units in a particular measurement standard, such as metric or imperial. This is a feature that has not been observed by the authors on any other web-based system. Additionally, although similar functionality has been seen in some other systems, it provides an ability to reason about whether it is possible to convert between two units—each unit is assigned a dimension (Davenport 2000*a*, Davenport & Naylor 2003); if the dimensions match, then the units are compatible. Each dimension is defined in terms of the others, except those for base quantities[1], and therefore the system could allow the user to enter combinations of units to build up a unit which is of arbitrary dimension, providing that both units share this dimension.

This project does not intend to start from scratch: there are a number of excellent unit converters available, and a selection of these will be examined as part of the Literature Survey. The analysis of these will then be used to guide the requirements gathering process, which follows the Literature Survey. The requirements will attempt to use the most relevant and helpful features from these other unit converters, and combine these with the full potential of using OpenMath as described above.

A system design will then be proposed, and analysis will be conducted to determine which requirements it meets, and whether there are any requirements it does not meet. The design will then be considered to evaluate whether the unmet requirements merit changes to the design. If changes are deemed necessary, these will be undertaken and the design re-evaluated until these are resolved.

[1] length, mass, temperature, time and current are defined at present.

Having completed this, the system will be implemented and tested, and the results analysed.

Finally, any insights gained through producing the system will be examined in the Conclusions chapter, along with suggestions for further work and resolving outstanding problems.

Chapter 2

Literature Survey

2.1 Introduction

In this literature survey, we intend to cover the background to the topic of units, and unit conversion, before looking into some existing implementations, making note of interesting features, and potential problems to avoid. Having completed this, we will turn to looking at using OpenMath objects as the storage for our converter, and discuss the merits and drawbacks of various options for this application, with the focus on graph-based implementations. Investigation of Algebra Systems and how unit abbreviations could be implemented in OpenMath will then be undertaken, followed by research on processing natural language and which OpenMath data formats are available. Finally, we will investigate usability and accessibility issues relating to web-based systems, and decide which of the relevant standards apply to this project.

2.2 Units

A unit of measurement is *any determinate quantity, dimension, or magnitude adopted as a basis or standard of measurement for other quantities of the same kind and in terms of which their magnitude is calculated or expressed* (Oxford English Dictionary 2007*b*).

Throughout history, different communities have used different systems for measurement, usually based on dimensions that were meaningful to them: for example, the length of a man's forearm was the definition for the cubit (Oxford English Dictionary 2007*a*). As communities began to trade with each other, it became increasingly important that they could understand each other in several respects, one of which was measurements. A farmer who wanted to sell his field might say that it was 25 rods long, while a potential purchaser of said field may know what he wants in furlongs. The two would then have to find a common measure, or they would be unable to continue.

For a long time, there *was* no common measure—for any dimension—and those that became "common" were not necessarily standardised. Over time, standards were implemented, and people could finally convert from one unit to another reliably.

Having said this, unit conversion has historically been a problem, even after standard conversion factors were defined. On many occasions, units have been incorrectly converted, due to human error or misunderstanding. More recently, a variant on this problem has appeared, where workers in different countries are involved, and they have different standards, for example US pint vs. Imperial pint. This project intends to allow the user to be explicit about which units are used, with no possible ambiguity.

2.3 The problem of conversion

There have been many famous examples where unit conversion was not undertaken, or where it was incorrectly calculated. The Gimli Glider (Nelson 1997, Williams 2003), as it

became known, was a (then) new Boeing 767 plane, which, during what should have been a routine flight in 1983, ran out of fuel just over halfway to its intended destination. The ensuing investigation established that an incorrect conversion had been performed, leading to the plane having less than half the required amount of fuel, because the aircraft was one of the first of its kind to use a metric measure of fuel, and the refuellers had used an imperial conversion instead of the correct metric one. In addition, although a second check was carried out between legs of the flight, the same incorrect conversion was used.

Large organisations such as NASA are not immune to such problems (Mars Climate Orbiter Mishap Investigation Board 1999). Software controlling the thrusters on the Mars Climate Orbiter was configured to use imperial units, while ground control, and the other parts of the space craft, interpreted values as if they were metric. This led to the orbiter entering an incorrect orbit too close to Mars, and being destroyed.

Even within individual units, there is scope for confusion. For example, the litre has been redefined several times, with recent versions known as the pre-1964 litre, based on the mass of water (Bureau International des Poids et Mesures 1901), and the post-1964 litre (Bureau International des Poids et Mesures 1964), where the litre was redefined in terms of length, as was its original definition in 1879.

2.4 Analysis of other Unit Converters

There are a number of unit converters already available. In this section, we will examine how they compare to each other, and what ideas we can take from them to use in the present project. In addition, we will attempt to show what advantage there is to using OpenMath—by showing deficiencies that exist in these systems that this project can correct.

Due to the abundance of existing online unit converters, we are conscious of the fact that we cannot possibly hope to survey them all. With this in mind, we decided to look at several already known to us that we had used in the past, as well as a limited sample from a web search. To choose which other converters to survey, we used the first 25 distinct results from a search on `www.google.co.uk` for "unit converter", and picked out the ones which had interesting or unusual features. Note that some sites, even though they were distinct, used identical systems to each other, so we chose, as far as we could tell, the site on which the converter originated. Having completed this process, we intend to look at the following unit converter systems:

1. `http://digitaldutch.com/unitconverter/`
2. `http://www.onlineconversion.com/`
3. `http://www.unitconversion.org/`
4. `http://www.convert-me.com/en/`
5. `http://online.unitconverterpro.com/`

6. http://www.knovel.com/knovel2/unitconverter.jsp
7. http://www.chemie.fu-berlin.de/chemistry/general/units_en.html
8. http://www.he.net/~seidel/Converter/
9. http://www.engnetglobal.com/tips/convert.asp
10. http://www.megaconverter.com/Mega2/
11. http://www.unitsconverter.net/
12. http://www.convertit.com/Go/ConvertIt/Measurement/
13. http://www.convertunits.com/
14. Google calculator—http://www.google.co.uk

2.4.1 Summary of features

Before we begin the detailed analysis, we will summarise the main features found in any or all of these unit converters. These converters are compared succinctly in terms of these features in the summary table, Table 2.1 on page 17.

Negative Conversions

If a negative number is specified to be converted, would the correct conversion occur?

Instantaneous

Does conversion occur as the value is typed in, or is, for example, a form submission required?

Multiple-at-once

Can one input result in many different output values displayed and updated simultaneously?

Categories

Were units split into different categories, and if so, how many were there?

Reverse conversion

Was it possible to modify the output value, and did this result in the input field being updated accordingly?

Natural Language interpreter

Was there a facility to accept typed user input of what units were desired, instead of, or in addition to, choice from a pre-written list?

If NL, does it support abbreviations?

If natural language input was supported, could abbreviations be used?

Conversion of compatible units only

Did the system enforce that incompatible units, such as a mass and a length, were not converted?

Suggestions provided based on user input?

This is also related to natural language support: as the desired unit name was entered, did the system suggest units for easy access?

Precision

How precise were results, and was this customisable? Note that this has little to do with accuracy—although an imprecise answer which has been rounded will also be slightly inaccurate. This may have been expressed in the number of significant figures or number of decimal places used when the converter returned results.

Query modification after conversion

Once the query has been submitted and the result(s) returned, is there a user interface feature that allows modification of the query to get a new result, whilst still able to see original result?

Copy-to-clipboard functionality provided

Is there a feature of the system's user interface to allow the result to be copied to the clipboard?

Calculations

Can conversions include a calculation, such as $2 + 3$?

Back-End Program

Does the system appear to call a back-end program, with the results displayed by the front-end?

Pop-up Window

Does the system offer the choice of opening the current converter in a pop-up window so that other pages can be used as well as the converter?

Unit Combinations

Can the converter process combinations of units, for example a measurement in yards, feet and inches?

We will now describe the converters surveyed in more detail. Where we have been inconsistent about the spelling of various units, for example metre/meter, we have used the spelling found on the particular site. Note that all these converters were accessed on 30/10/07, and worked as described.

2.4.2 http://digitaldutch.com/unitconverter/

This site splits conversions into several categories and then provides a drop-down list of units for the category of choice. This is useful in several ways. It means that the user cannot enter "nonsense" queries such as "1 kg in metres", and also avoids the difficulty of parsing the user's natural language input.

This site works quickly, due to the use of JavaScript (JS) to perform the conversion, but this causes a few problems—if the user deletes the text in the input box, the value in the output box will be updated to show "NaN"[1], which is not very useful. The site supports reverse conversion—if you modify the value in the output box, the input box is updated accordingly. Interestingly, invalid numbers such as "2.3.4" are treated as if only the value

[1] although not explained on the site, this means "Not a number'

up to the second decimal had been entered—i.e. "2.3" in our example. Another problem with the site—other than the "NaN" display, as previously mentioned—is that inaccurate, or imprecise, measurements can be returned. For example, entering 10 cups in litres gives 2.365882. However, modifying the output value to 2.3658824 leaves the input at 10, and then removing the 4 on the end, to return to the original output, changes the "input" value to 9.999998, when it should have been 10 going on the initial result. This may be due to the user-modifiable option to set how many decimal places are used, or JS rounding errors.

It is not possible to perform calculations in the input.

The system also includes common prefixes built in—you can choose to convert into Newtons or kilonewtons as separate items. Obviously for some of the conversions, compound units are allowed.

Overall, this is a quick and user-friendly converter, which suffers from a few minor drawbacks.

2.4.3 http://www.onlineconversion.com/

Contains a work-in-progress natural language-interpreting converter, which works if examples are followed: "5 feet to metres" works, but "5 feet in metres" gives an answer in m^3—although this appears to be because "in" is interpreted as inches, so three length dimensions becomes m^3. A more traditional selection process is also available, which works via form-based input. This can be slower than the one in section 2.4.2 (page 9) as the conversion is not immediate. The form will remove any non-numerical characters from the input box before processing—leaving any decimal points or minus sign. Due to the sheer number of units available, this site has many categories and subcategories which the user must peruse before locating the desired unit. Thus, converting units is quite a slow process: once the site has been reached, several steps have to be undertaken before getting to the correct page. The natural language interpreter of course should deal with this. If a category offers a large number of choices, they are split up, on a separate page, into "common", "all", or "metric". Overall, this is a reasonably good converter, which has a natural language-parsing feature under development, but is let down by the sheer number of menus that need to be traversed.

2.4.4 http://www.unitconversion.org/

In the absence of a natural language interpreter, this site has a handy feature that bypasses the problem observed in section 2.4.3 on page 10—you are able to type the measurement unit you want into a box, and as you type each letter a list updates suggesting which unit you might wish to use with a link to the respective page. This goes some way to avoiding the categories and then subcategories problem, but is slightly marred by inconsistencies—it recognises "meter" as a unit of 'general' length but then also "metre" as a unit for wavelength only—the page for which only provides conversions between wavelengths and

frequencies. The site also offers links to category pages, with additional specific links to go to a page of information on that unit, the latter page containing yet more links to convert directly from that to a different unit. It might however have been more useful for the link to take the user into the category page with the specified unit selected in the "from" box.

The website appears to offer many units to convert between, but in fact many are just multiples of others—for length, the following forms of meter are available: "meter", "exameter", "petameter", "terameter", "gigameter", "megameter", "kilometer", "hectometer", "dekameter", "decimeter", "centimeter", "millimeter", "micrometer", "micron", "nanometer", "picometer", "femtometer", "attometer", i.e. the range 10^{18}m to 10^{-18}m, including two options for 10^{-6}m. The site allows reverse conversion, and does not fall into the same pitfall as the converter in section 2.4.2 (page 9): working with higher precision than that in which the converter returns results does not lead to inconsistent results—while still allowing instantaneous conversion, as the user types. The precision is configurable via a drop-down list. Also, a calculation can be entered into the box, such as 5*3+2, and will be duly converted as if the user had typed the value, 17 in this example. Additionally, if the input box is blanked out, the output is also blanked, rather than filled with the unhelpful `NaN`. The site features an "expanded" version of the converter which presents boxes for all the units of that measurement type available. When the user types a number into one of the boxes all the other boxes are recalculated—although it can be hard to find both units as it is quite possible they will not fit on the screen at the same time. Overall, this is a quick and precise unit converter, with several useful features—even including an information page about each of the units.

2.4.5 http://www.convert-me.com/en/

This site works in a similar way to the "expanded" feature described on the converter in section 2.4.4 (page 10), except that the conversion does not happen as an input box is modified—the user has to click calculate, or press <Enter>. The site supplies a large range of units to convert between, from a variety of categories. On each category page a series of labelled input boxes is displayed—such as one labelled km/s^2 on the Acceleration page. The value is typed in, and the form submitted, whereupon the other boxes are updated to reflect the converted value. This is useful if it is desirable to see what a value in one unit is in many others, but can be a bit of a problem if only one other unit is required—the conversion is quick so it does not take perceptively much longer to convert all the values, but if the two units required are far apart on the page it can be tricky to find them. The units are split into sub-sections on the page; for example Metric, US and Imperial; and all sub-sections get updated at once. On some pages, the substance being converted is required, in case it affects the result. The number of significant figures is configurable via a drop-down box. There are not that many units available on this site, especially when you take into account that 7 of the 12 metric "mass and weight" units, for example, are multiples of the gramme. Each unit listed is a link for writing the desired "from" value in to a text box. Clicking the link, however, takes the user to a new page which expresses the quantity 1 of that unit in all the other units that can be calculated, which may be useful.

Overall, this site is not particularly helpful; there are not that many units available, and they are widely spaced apart, with big banners separating subsections of the page. This means that, although the conversion happens for all the units, a lot of scrolling has to be performed if the units are not in the same sub-category.

2.4.6 http://online.unitconverterpro.com/

This site offers 76 categories of unit. It appears to work with 11 significant figures, but there is no way to customise this. Unusually, simple calculations are allowed. Although it does take a few mouse clicks to get to the correct page, the site has a potentially useful feature in that it offers a "common units" page, a "metric units" page, and an "all units" page for all categories. This means that if it is known that the units required are both on the "common" page, or both metric, that one can be used directly. This leads to the obvious question of what should be classed as common, however. On the "all" page, as with the converter in section 2.4.4 (page 10), there are many multiples of the meter. The site offers the option of sorting the list of units in either Alphabetical order or Logical order, where the list for length starts with "exameter" and goes down through "meter" to "attometer", followed by "angstrom" (sic), and "fermi"—in addition to its equivalent, the femtometer. We are not sure which is more useful—in Logical order, more than a screenful[2] is taken up with multiples of the meter, and, as it is unlikely that the user wants to convert between two of these, this is a lot of wasted space. In alphabetical mode, it is easier to know where to look for the unit you want, but there are quite a few obscure ones that may get in the way most of the time. The "common units" page is a lifeline here, but, again, several multiples of the meter dominate the options for length. In summary, this site is quite useful for a wide variety of conversion types, and offers drop-down list access to the "all units" page for each category—as well as a link to a page with the three options.

2.4.7 http://www.knovel.com/knovel2/unitconverter.jsp

This site offers a drop-down list of categories and for each populates a pair of drop-down lists with the units available. The number of significant figures used can be specified in another drop-down box, and text boxes are provided for input and output. The site has nearly 90 categories from which to choose. Reverse conversion is not possible, but this is clearly indicated with an arrow showing the direction of conversion. The User Interface is quite slow and unfriendly—it requires a lot of mouse movement to select the units and submit the query—you cannot just press <Enter> once you are satisfied, but this site does offer an unusual feature of having a button to facilitate the copying of the result to the clipboard. In summary, this site has a useful feature and has a lot of conversion categories, but is frustrating to use.

[2] the box displays 14 items, with a scroll bar

2.4.8 http://www.chemie.fu-berlin.de/chemistry/general/units_en.html

This site basically provides a web front-end for the UNIX `units` utility. It provides boxes for input and output, and you are free to enter numbers into either. Then two drop-down boxes offer all the different available units, so it can take quite a while to locate the correct one. The user can then click submit, or press <Enter>, whereupon they are taken to a different page with the result on. Both the forward and reverse conversion factors are displayed as well as the result itself. As the "To" unit could have a multiplier value (e.g. selecting `mile` in the "from" list and `yard` in the "to" list, and entering `1` in the "from" text box and `5` in the "to" text box), strange results such as `1 mile = 352 5yard` (that is, 1760 yards) can be obtained. The user interface is not that helpful because you cannot modify the query whilst still looking at the previous result—except by modifying the address bar itself—as you can with many of the others. Also, the user interface (UI) makes no attempt to stop you from converting nonsense; for example, with "miles" to "ounces", a conformability error is returned, after submission, from the units program:

```
conformability
8.046720e+03 m
2.834952e-02 kg
```

A slight oddity occurs if you try to convert 0 of one unit to another—for example, entering 0 yard to feet gives:

```
0 yard = 3 feet
```

which is clearly untrue. In addition, if you try to convert a negative value, any negative integer will give the output for +1 and a negative decimal will give an error "cannot recognize .". This is thus the only converter of those surveyed that lacks support for negatives.

This site, however, does provide an insight into calling a separate back-end executable, as is the intentions of this project. Interestingly, the page appears to parse the output, with parts of the output from the `units` program split under different headings.

2.4.9 http://www.he.net/~seidel/Converter/

This page presents the user with a box to type the input value in, and a drop-down list of all the supported units. The user then submits the form, and is provided with a list of results for the units available for that type—choosing inches on the first page will result in only length measurement results being presented on the second. Like the converter in section 2.4.8, it is not possible to modify the user input whilst looking at the results—although it is shown on the output page. An oddity is that there are more units on the list than will be converted to, and if you try to convert from meters, it only displays the result for centimeters, but if you convert from centimeters you get results for inches and several obscure/foreign units. In summary, not the most useful of converters, but it does say on the main page "Well it works, but it could be better".

2.4.10 http://www.engnetglobal.com/tips/convert.asp

This site provides a search box for units, which then provides links to the conversion pages it thinks the user desires. For example, entering "feet" brings up a variety of links—"Acre-feet (Volume), Board feet (Volume), Cubic feet (ft^3) (Volume), Cubic feet/hour (Volume Flow), Cubic feet/minute (Volume Flow), Cubic feet/second (Volume Flow), Feet (International), (Length), Feet (US, Survey) (Length), Feet/$second^2$ (Acceleration), and Square feet (Area)". Picking any of these takes the user to a page with all the available units for that type—for example choosing "feet (International)" takes the user to the metric length[3] page, which provides a long list of units—typing a value for one updates all the other values on the page. However, despite being aimed at engineers, the precision available is quite low, with each unit having a minimum non-zero value of 1e-7—any value that is closer to zero than this than this is displayed as 0. In summary, we feel this is not a very good converter; it is mainly let down by a lack of accuracy and precision. The search facility is useful.

2.4.11 http://www.megaconverter.com/Mega2/

This site provides one of the bigger lists of categories of any of those surveyed—45 in total. The site has an interesting and quite helpful interface, although much of it is unrelated to the unit conversion itself. On a UI front, it does have an option to open the current conversion in a pop-up window, so that if you need it but also need to go to a different page, you can do both. Once the user has picked a category from a drop-down list, they are taken to a page with a list of all the available units for that category, in the same manner as many of the other sites. In summary, this does have some nice UI-related features, and a wide range of categories.

2.4.12 http://www.unitsconverter.net/

This site is noteworthy because, although it only uses a small subset of units, it can work with combinations of those units. The site presents a list of categories with a selection of relevant units in each. The novelty comes because it provides boxes for (in length) Feet, Inches, Mils, which the user can either enter values into individually or in combination and it will correctly convert to the other units—metres/centimetres/millimetres (both separately and individually), miles, kilometres, microns, nanometres, Ångstroms. Updating one box and submitting updates the rest. This feature is quite useful, particularly for imperial conversions (in metric, the differences tend to be just powers of ten), and this would be even better if it could be extended to a wider range of units.

[3] a misnomer; all the available lengths—metric or otherwise—are present

2.4.13 http://www.convertit.com/Go/ConvertIt/Measurement/

This site provides two text boxes in which the user types the units they wish to convert from and to, and also, in the "from" box they type the quantity as well. This is then processed to work out which units were intended—if the user typed "3 yd" in the "from" box, the page will process this as 3 yards. This natural language system works quite well, and if it does not recognise the unit(s) specified, a warning message is provided explaining this. A particular oddity of the site–it seems to be a feature, rather than a bug!—is that you can convert incompatible units.

To demonstrate this, the input "From: 2 feet To: kg" yields:

```
2 foot (length) TO kilogram (mass) = 0.6096
meter / kilogram (reciprocal of mass to
length)
```

As can be seen, the page has correctly identified the type of the units involved, but has proceeded with the calculation regardless. we can think of no use for this feature, but the fact that it would be so easy to prevent once the two unit types have been established leads us to assume it has been left in for a reason! In summary, this site offers a useful insight into working with name abbreviations, but also explicitly allows conversion of incompatible units.

2.4.14 http://www.convertunits.com/

In a similar way to the converter in section 2.4.13, this site detects and displays the type of units being converted. However, *unlike* the aforementioned converter, this site *disallows* converting incompatible units with an error. It then proceeds to provide an explanation of the two units and why they were deemed incompatible. In addition, combinations of units, such as "5 feet 2 inches", can be converted correctly—although only up to a maximum of 2: attempting to convert "1 yard 2 feet 3 inches" into "metres" results in an error. Although in the first instance the conversion does not happen instantly, as the user must submit the form, after submission the user is taken to a page where the units are fixed but the values can be changed, and here the conversion happens instantaneously. An interesting quirk of the site is an "I'm feeling lucky" feature, which provides two random (compatible) units with text boxes into which you can type a value and the other box is populated with the converted value once focus leaves the entry field. The home page also provides 20 links to random compatible conversions which work in the same way as the "I'm feeling lucky" feature. However, it is unlikely that this would ever be needed for serious use—if it is desired to convert units, presumably it is already known which units are required. The feature of explaining why the units could not be converted is very useful, and if the user types a unit that is not recognised, the page attempts to suggest similarly-spelt ones, in case they mis-typed. In summary, we think this is a very useful converter with quite a few useful features that are not present in other converters.

2.4.15 Google calculator—http://www.google.co.uk

It may be worth mentioning at the outset that, as `http://www.google.co.uk` is predominantly a search engine, it should not be expected to have the advanced conversion capabilities of other, dedicated, converters. However, the conversion part of the functionality is fairly advanced—it takes a natural language input, for example "33 km/litre to mpg", and outputs the result, along with the option of searching the Web for the user's request. It can use a variety of names for most units on offer—km, kilometre, kilometer, kilometres, and kilometers, but NOT kms, which is not a valid abbreviation, are all recognised to mean kilometre. Other features include that arbitrarily complex calculations (for functions that are recognised) can be performed—"$\ln(e^5)$ microns to miles", although perverse, produces the correct output. Also, combinations of units, such as "2 feet 5 inches to metres", are correctly processed. For units that are not recognised, or incompatible, the converter functionality is not displayed, and a normal web search is executed. One disadvantage of this approach is that there is no publicly documented list of the units recognised, so it can happen that when a conversion is desired, only a web search takes place. In summary, this site is useful as it combines several facilities into one, that none of the other converters have, but not knowing what units are available can be a problem.

2.4.16 Summary

The main features of all these converters is summarised in table 2.1 on page 17.

	Feature															
	Neg	Ins	Mul	Cat	Rev	NL	Abb	Con	Sug	Pre	Mod	Cop	Cal	BE	P-u	Com
2.4.2	✔	✔	✗	12	✔	✗	N/A	✔	✗	1–15dp	✔	✗	✗	✗	✗	✗
2.4.3	✔	✔	✗	27	✗	✔	✔	✔	✗	10dp	✔	✗	✔	✗	✗	✗
2.4.4	✔	✔	✔	79	N/A	✗	N/A	✔	✗	0–15dp	✔	✗	✔	✗	✗	✗
2.4.5	✔	✗	✔	22	N/A	✗	N/A	✔	✗	1–7sf	✔	✗	✗	✗	✗	✗
2.4.6	✔	✔	✗	76	✗	✗	N/A	✔	✗	11sf	✔	✗	✔	✗	✗	✗
2.4.7	✔	✗	✗	87	✗	✗	N/A	✔	✗	1-16sf	✔	✔	✗	✔	✗	✗
2.4.8	✗	✗	✗	✗	✔	✗	N/A	✔	✗	7sf	✗	✗	✗	✔	✗	✗
2.4.9	✔	✗	✔	✗	N/A	✗	N/A	✔	✗	15sf	✗	✗	✗	✔	✗	✗
2.4.10	✔	✔	✔	21	N/A	✔	N/A	✔	✔	7dp	✔	✗	✗	✗	✗	✗
2.4.11	✔	✔	✗	45	✔	✗	N/A	✔	✗	5dp	✔	✗	✗	✗	✔	✗
2.4.12	✔	✗	✔	10	N/A	✗	N/A	✔	✗	10dp	✔	✗	✗	✗	✗	✔
2.4.13	✔	✗	✗	✗	✗	✔	✔	✗	✔	15sf	✔	✗	✗	✔	✗	✗
2.4.14	✔	✔	✗	✗	✔	✔	✔	✔	✔	16sf	✔	✗	✔	✗	✗	✔
2.4.15	✔	✗	✗	✗	✗	✔	✔	✔	✗	9sf	✔	✗	✔	✔	✗	✔

Table 2.1: **Summary of unit converter features**.
Key: Neg—Negative, Ins—Instantaneous, Mul—Multiple-at-once, Cat—Categories, Rev—Reverse conversion possible?, NL—Natural Language interpreter, Abb—If NL, does it support abbreviations?, Con—Conversion of compatible units only, Sug—Suggestions provided (based on user input)?, Pre—Precision (range or specific), dp—decimal places, sf—significant figures, Mod—Query can be easily modified after conversion, Cop—Copy-to-clipboard functionality provided, Cal—can process calculations, BE—appears to call back-end program, P-u—option to open conversion in a pop-up window for use with other sites, Com—can process combinations of units at once. N/A—not applicable for this converter, for example the "reverse" of a conversion where Mul is true is undefined.

2.4.17 Conclusion

Although there are a great many unit converters available, they tend to work in a similar way, with just a few exceptions. A common drawback of all of these featured converters is that, while some of them have a great many conversion factors built in, none has the option of extending the range of conversions by adding new units. A unit converter with many units tends to be more useful than one with few, although if they are difficult to find this may not be the case. It is worth considering the number of units available in a particular converter—although how easy the units are to access should be a part of this—but it must be borne in mind that there will always be a unit that a given user might like to use which is missing.

With regard to speed, many of the converters utilised JavaScript to facilitate conversion as the user types. Others would perform the conversion after the focus had left the text box, or only after the request was submitted. If the converter did not allow natural language as the input, the available units were almost invariably split into categories, of varying number.

Many of these sites merely stored a conversion factor to a "standard" unit—as an example, for length, all conversions might be stored in terms of Plancks, as the smallest value, or metres, as a common value—so that it is simpler to write but slightly inefficient—all calculations go via this standard unit rather than directly between the two chosen units. Storing in this way presumably confines the converter to units which have a constant multiplicative factor—ruling out temperature and logarithmic units (e.g. power/volume intensity). This is because there is no information stored about how the number specified should be used, and usually only one value can be stored per unit—in the drop-down list method of many implementations—so the system has to have hard-coded into it *how* the value will be used.

With very few of these converters is it possible to find out exactly which conversion factor they are using—for example, do they mean US pints or imperial pints? Even if they have specified the particular version of the unit involved, there is usually no direct record of the conversion factors used. It is possible to find out, by converting 1 into a metric unit for example—an imperial pint will come out as approximately 568ml, whereas a US pint would come out as some amount less, depending on whether the liquid pint or dry pint were being used.

All of the converters used a form-based input system. This seems to be a sensible way to accept the user input. Most of them allowed the user to easily modify and resubmit their query—with only `http://www.chemie.fu-berlin.de/chemistry/general/units_en.html` and `http://www.he.net/~seidel/Converter/` is this not possible. Although not mentioned, all converters displayed the user's input as part of the output—regardless of whether modification of this value was possible.

Most of the converters allowed the use of compound units to some extent—either choice from a list or typing in the unit.

Very few of the converters surveyed appeared to call a back-end program—most seemed to store the conversion factors in the form of JavaScript, and use these during the conversion. This allowed much faster conversion, as the page was self-contained, and another one did not need to be loaded, at the expense of the page being larger in size, so initial download time would be increased.

With the converters that had categories, while being more useful than being presented with a huge list of all available units, some did suffer from having too many categories available, and maybe would have benefited from a "common categories", in addition to the feature that several did have of a "common units" page within a category.

OpenMath's benefits

How can a system using OpenMath offer advantages over the presented systems?

OpenMath's extensibility is the main advantage—a user could submit new units to the OpenMath Society, or use a "private Content Dictionary" (Buswell, Davenport, Carlisle & Dewar 2004) which only they have access to. This would allow temporary addition of obscure units for a particular application, or adding new units that become more common so that the entire OpenMath community can use them. The central repository of Content Dictionaries (CDs, see section 2.5), and thus unit definitions to be used, simplifies the task of determining which conversion factors have been used between any given units. Of course, if the user has provided a CD of their own, they will have a copy of it so would know what conversion has been used.

2.5 OpenMath (and MathML)

OpenMath is a standard for storing mathematical semantics (Buswell, Caprotti, Carlisle, Dewar, Gaëtano & Kohlhase 2004), which at the time of writing is Version 2.0. Its main aim is to allow transmission of mathematical semantics between programs, or over the Internet, for example. OpenMath objects (the way the semantics are stored) can also be stored in databases.

To ensure the semantics are not lost, or, worse still, misinterpreted, upon transmission between programs, OpenMath defines a set of Content Dictionaries, which contain various symbols and definitions. For example, there is a Content Dictionary with a definition for the symbol π (approximately 3.14159), and various mathematical operators, such as times, for which there are several distinct options. Content Dictionaries can be added to the collection by users, but they must be submitted for approval to the OpenMath Society. If a user wishes to refer to the number π, they would refer to the specific reference to it in the correct Content Dictionary—in this case, the Content Dictionary is `nums1`, and the symbol is called `pi`. If it is desired to use a commutative n-ary times function, the one in the Content Dictionary `arith2` would be used.

How is this relevant to a unit converter? Several Content Dictionaries of units were proposed in Davenport & Naylor (2003): `units_metric1`, `units_imperial1` and `units_us1`. These contain definitions of many common units covering a variety of dimensions—metric (SI) units are contained in `units_metric1`, for example. The dimensions themselves are defined in the Content Dictionary `dimensions1`. This document suggests using the "usual" times operator (that stored in `arith1`) to represent a number in a particular unit—i.e. storing the value as the number multiplied by the unit, with the unit following the value to which it refers. The suggestions for unit "implementation" in OpenMath are stated as being based on those used by a complementary mathematics display language, MathML—although not blindly; where the authors believe MathML has some deficiencies, these have been stated and corrected. This document also specifies a reasonable way of connecting a prefix to a unit.

2.6 Graphs

A major section of this project involves getting from one unit to another. In some cases this is trivial—for example, the foot is defined in terms of the metre, so it should be fairly straightforward to switch between the two. The difficult case arises when the two units are non-adjacent—i.e. the definition of one involves at least one intermediary before reaching the second. As an example, the yard is defined in terms of the foot, which is defined in terms of the metre, so if it was desired to convert yards to metres the definition of the foot would also be used. The obvious way to deal with this problem appears to be to turn the system into a graph-traversal problem. This would involve loading in all the definitions somehow and mapping the relations to a graph.

2.6.1 Storage

Bell College (2006) describes the data structures that can be used for the nodes and edges of a graph (albeit in Java), so we intend to base our design on this. Skiena (1997) agrees that a list, rather than a matrix, would be the most appropriate data structure for a small graph, as we will have.

We think that an undirected, unweighted graph for each dimension of unit should be created, storing the conversion factor between two units as a feature of the edges between nodes—it would not be the weight, as we do not believe it is worth setting different weights for different calculations. If we end up using an algorithm which requires graph weights, we will set them all to 1, but as most of the conversions will be simple multiplies, the cost of calculating the exact weighting would be far in excess of the gain from doing so. We choose to use an undirected graph because the conversion factor in one direction will simply be the inverse of the conversion factor in the other direction, and the OpenMath object stores the conversion factor as well as how it relates to both units. The dimension for each unit is stored in the Small Type System file for the CD that contains the unit.

2.6.2 Traversal: Shortest Path

A large number of papers have been written on the subject of finding paths round graphs, describing algorithms of varying efficiency and function. As this is a time-consuming task, we intend to have a program that runs once to load all the relevant online CDs and build up a list of units and shortest paths between them. This could then be loaded and used as required, without having to work it out every time. The only times these data would require updating would be when the CDs that are needed are either updated or new ones are added, or, in a related case, when the user supplies their own. However, we feel this will be significantly quicker than calculating the best path each time the user wishes to convert units.

There are several kinds of shortest ("best") path algorithm available: those that find the shortest path between two nodes, those that find the shortest path from one node to all the other nodes, and those that find the shortest paths between all the nodes. There are variants of these for both weighted and unweighted, as well as directed and undirected, graphs. As we have chosen to use an adjacency list as the predominant data structure, rather than an adjacency matrix, this rules out those algorithms that demand matrices (Seidel (1992), Zwick (1998) being examples). The most famous algorithm in the area is the first found in Dijkstra (1959), but this algorithm only finds all the paths from one source node. Finding all the paths using Dijkstra (1959) thus involves running the algorithm N times for N units, with a resulting minimal complexity of $O(N^2 \log N)$, or at most $O(N^3)$. As we envisage calculating all the paths to all the nodes first, and storing this in a file, it may be worth finding a more efficient algorithm for finding all the paths at once. Johnson (1977) describes just such an algorithm for finding the shortest path between all nodes, but goes on to cite Wagner (1976) as a better algorithm when the weights are low (as previously stated, we anticipate the weights all being set to 1). The graph will be what is known as an "Edge-Sparse graph", as there will be few edges between units (approximately $N - 1$ edges, for N units—this is based on the expectation that the graph will mostly have a star-like structure with several nodes coming from one central node—rather than more of the nodes being connected to each other; each unit definition will be defined in terms of exactly one other unit), and using Wagner's algorithm would be faster than Dijkstra's famous algorithm in this case (Wagner 1976). However, Wagner's algorithm is significantly more complicated to implement than Dijkstra's.

Dijkstra (1959) also contains a second algorithm, which could be used to find the shortest route without having to load all the nodes of the graph—starting from the source node, only those nodes required until the destination node has been found need be loaded. This could speed up the processing, or traversal, of the graph, and also finds the shortest path.

Zwick (2002) (an update to Zwick (1998)) provides a different approach—attempting to find the shortest path that additionally uses the fewest edges. This is unnecessary in our case, due firstly to the unweighted nature of the graphs we will be using; any shortest path will use the fewest possible edges, and secondly because of the added complexity involved. In addition, it uses matrices.

Chan (2006) and Chan (2007) describe faster algorithms, but the algorithms in the latter are more efficient for dense graphs, and our graphs will be very sparse. Chan (2006) points out that using a breadth-first search has complexity $O(mn)$, where m is the number of edges and n is the number of vertices. As we are anticipating $m = n - 1$ (or certainly $O(n)$), this results in $O(n^2)$. The breadth-first search seems to be one of the simpler algorithms to implement, and gives the optimal result when the weights for each edge is the same, as it will always find the route encompassing the fewest edges possible. However, this algorithm is designed to find a route between two nodes, which is less efficient if we wish to find all the routes between all the nodes, and as such it may be better to only consider this algorithm if we decide to perform the lookup each time the user enters a query. On the other hand, if we do have to implement an algorithm, this is a fairly easy one to implement, and because we will be using such a small number for n, the complexity is not really going to be a huge issue; the time taken to implement a particular algorithm will be more of a deciding factor. we will investigate these further during the design phase of the project; a highly-optimised algorithm is probably more effort than it is worth, as none of our graphs will have many nodes or edges.

2.6.3 Implementation

We could write our own graph-related data structures and algorithms. However, as described in de Halleux (2007), there are freely available libraries for .NET for creating and traversing graphs. QuickGraph is a free, open-source set of libraries written in C#. It includes functions for several shortest path algorithms, including Dijkstra's algorithm. As stated previously, we would prefer to use a different algorithm, as Dijkstra's is not the most efficient, but we may choose to implement using this method, then change to a more efficient algorithm if necessary once the system is working.

2.7 Algebra Systems

Another major part of the system is working out how to carry out the conversion once a path through the graph has been found. For example, if the conversion required was from yards to metres, we would first determine that the best path was from yards to feet to metres, and therefore we would have to apply the conversion for feet to the value in yards, before applying the conversion for metres from feet to the result of that[4]. This would involve computing the result of applying an arbitrary piece of algebra to the input value. However, as most conversions are just a single multiplier, we do not think it is necessary to use one of the many available algebra systems—they are overkill. Some conversions require more than just a multiplier—for example, temperature conversions are often a multiplier and an addition. Another class of conversions, for example, amplitude to volume in decibels, requires a logarithmic calculation, but again, it should not be too difficult to implement

[4]Or, of course, combine the two formulae appropriately and then apply the combination

these few operations (add, multiply, log) ourselves, without resorting to an algebra system.

2.8 Abbreviations

As units can be referred to in general by potentially several different names (for example, it is unusual to write "kilogrammes", but far more usual to write "kg", "kgs", or "kilos"—though kgs is incorrect, so whether it should be included or not is a different issue), and as we intend to be able to interpret all these abbreviations/symbols correctly, we will need some method of associating them with the OpenMath unit. When choosing our approach, however, we need to bear in mind that more units can be added, and abbreviations could be added or taken away. An example of an abbreviation being taken away is the litre, which can, at the time of writing, be described by the symbol L or l, but it is envisaged that only one of these will be used in the future (Bureau International des Poids et Mesures 1979). It makes sense to define a new Content Dictionary (or several) to contain these alternatives (presumably using some sort of equality operator). Davenport (2000*b*) suggests what seems a sensible way to do this. However, as stated in Buswell, Caprotti, Carlisle, Dewar, Gaëtano & Kohlhase (2004), to remove a symbol from a non-experimental Content Dictionary, the author is required to create a new CD, so that users of the old one do not have problems. Davenport (2000*b*) demonstrates that it is a reasonable idea to create at least one new CD for these abbreviations[5], and explicitly advises against putting too much into a Content Dictionary, as CDGroups can be used to group the data as required. We intend to group them in separate metric/imperial/etc. groups, and might possibly have a separate CD for each unit, as some will have many aliases. Once we have written these CD(s), they will be submitted to the OpenMath Society, via Professor Davenport.

2.9 Processing Natural Language Input

To process natural language, the input will be split into words. A parser will then determine which parts of the string are the "from" units and which are the "to" units, and these units will then be looked up. As in the case of this project this will be fairly trivial, we have chosen *not* to look further into this area.

2.10 OpenMath Data formats

OpenMath has two data formats (Buswell, Caprotti, Carlisle, Dewar, Gaëtano & Kohlhase 2004), which are discussed in this section.

[5]there is a new piece of semantics to convey, which can be written informally, and there is a motivation for doing so

2.10.1 Binary

This format is designed to be small, fast, and efficient, with large amounts of data in mind (Buswell, Caprotti, Carlisle, Dewar, Gaëtano & Kohlhase 2004). It is, however, not a requirement that an OpenMath application can process the binary format, and as such the majority of OpenMath data files in the public domain are in the XML format. Therefore, we do not think it is worth the effort of implementing this format, as we *must* implement the other in any case.

2.10.2 XML

OpenMath data can also be stored in an XML format. It is far easier for the human user to supply CDs in this, and those available online are generally in this format. This is a direct consequence of correct XML format processing being a minimum requirement for an "OpenMath compliant application" (Buswell, Caprotti, Carlisle, Dewar, Gaëtano & Kohlhase 2004), and as such, our program will need to be able to parse XML to be classed as an OpenMath-processing program. In addition, the XML format is intended for the Web, and our system will be web-based.

Parser

The .NET framework—at this stage we are anticipating use of C#, a part of the .NET framework—contains a set of classes to facilitate this, and we intend to make use of them, as there is no point in writing a new XML parser. However, there are alternatives available for other languages, such as Java or C if we require them.

Binder

It may well be worth using an XML binder to load the XML data files into OpenMath objects internally defined in our program. Depending on the language we choose, this may require using a third-party XML binder or one supplied as part of the language framework—.NET has one. In Bourret (2007), it is explained that data binding allows the application to use terms that are meaningful to it, rather than use the DOM, or XML document structure. This documents suggests Dingo as a possible XML Binder for .NET, which has some features that the built-in XML binding system for .NET does not have. However, we are not sure we will require these advanced features, so will focus on using .NET's built-in Binder. We will bear the alternatives in mind if we find we require them, however.

2.11 Usability and Accessibility

We want our unit converter to be usable for as wide a range of people as possible. As it will have a web-based front-end, it is already potentially *accessible* for a large number of people (anyone who has a web-browser), many of whom may have disabilities of some kind. Therefore, to ensure that it is *usable* by these same people, we will need to adhere to usability and accessibility guidelines/rules for web pages. The guidelines are specified in World Wide Web Consortium (1999), with additional notes in World Wide Web Consortium (2000*b*) and World Wide Web Consortium (2000*a*). As we have no intention of using sound as a primary method of conveying information, we do not need to do anything extra to cater for those who are either deaf or hard of hearing. However, as a given user may be visually impaired to any extent (including colour deficits), we may opt to utilise sounds to guide them through the interface, as well as being careful about our choice of colours. In terms of accessibility, we will have to use "standard" web code, that will work correctly and as expected, in as many browsers as possible. We intend to conform to at least the "double-A" standard described in the above documents, with possible conformance to the "triple-A" standard. We believe the guidelines that will be applicable to our system are as follows: 1 (possibly—if we use images), 2 (probably), 3, 4 (part 3), 5 (possibly), 6 (possibly—depending on the technologies we use), 7 (if we use, for example, a suggestions box), 9 (specifically parts 3, 4 and 5), 10 (specifically parts 2, 4 and 5), 11, 12, 13 (possibly) and 14. Therefore, we believe that only Guideline 8 is wholly inapplicable to our system, the reason being that we do not envisage having an embedded UI within the front-end.

2.12 Summary

Through this Literature Survey, we have gained an insight into a large variety of topics which all relate to our proposed unit conversion system in some way. We intend to use the comparisons we have made of several of the unit converters currently available to decide which features should appear in ours, particularly for the front-end. Other sections have shown how to implement the back-end part of our system, which has changed our view of the system and will go on to affect the design and requirements of the system.

Chapter 3

Requirements

3.1 Introduction

This chapter will present the User Requirements, obtained in a preliminary stage of the project, before expanding these into the System Requirements. Following these, a brief discussion of several of the requirements will be undertaken, to clarify or expand on aspects of importance. As the system does not have a specific user in mind, we have elected to use inspiration from the unit converters examined in the Literature Survey as the basis for many of the requirements, with the addition of OpenMath-related requirements.

3.2 Definitions

built-in unit When a unit is described as being built-in to OpenMath, this means that the user will not need to provide a definition for the unit for the system to understand it.

Content Dictionary These are the data files used by OpenMath which will store the definitions of the units.

Shall This feature must be implemented for the system to work.

Should If this feature is not implemented, the system as a whole will work, but with limited functionality.

May This is an optional feature that adds to the user experience, but is not essential.

3.3 User Requirements

This is the preliminary set of User Requirements, taken from the Project Proposal (in Appendix G), with additional requirements based on research since.

1. **The system shall correctly convert between compatible units built into OpenMath and return a result within 10 seconds. The system shall find the unit definitions without user intervention.**
   Rationale: The main source of units will be Content Dictionaries on the OpenMath website, and the system must be able to find and use these. The user will probably not know where the unit definitions are found, and as such will not be able to tell the system where to look. The time limit is imposed so the user does not waste time using the system—for built-in units the conversion should be rapid.

2. **The system shall correctly convert between compatible units not built into OpenMath. If the user requests a conversion where one of the units is not known to the system, it shall notify the user and ask for a CD containing the definition.** Having received this, the system shall respond to the

user's conversion request. This whole process (discounting time taken for the user to give the file to the system) shall take no longer than 20 seconds.
Rationale: This will allow the system to be extensible. The user needs to know that a unit was not recognised by the system, or the system will be fairly unusable. Equally, the user must be allowed to provide new definitions to the system. The time limit imposed is due to the reasonable expectation that the system will take longer to read the input, establish that a unit was unknown, notify the user, read the new file(s) and provide the result than just reading the input and providing the result. Twice as long is a reasonable length of time, though shorter is of course preferable.

3. **The system should allow the user to provide a new valid Content Dictionary to the system when a conversion is not in progress, which it shall process and use correctly. In addition, the system shall be able to use any new ones which appear on the OpenMath website.**
Rationale: This will allow the system to be extensible, and users will be able to convert between any units they like, providing they, or another member of the OpenMath community, have written a valid Content Dictionary for it.

4. **The system may be able to understand a conversion request in plain English (e.g.** `5 miles in km`**).**
Rationale: Typing in this way is more natural for the user to express their request. However, a two input approach is also reasonable.

5. **The system shall flag up an appropriate error if the user attempts to convert between two incompatible units.**
Rationale: This is more useful than trying the conversion anyway, and more informative than just refusing to do it.

6. **The system shall be able to convert to appropriate units in a given measurement standard (e.g. converting the request** `miles in metric` **would (probably) choose km as appropriate).**
Rationale: Often a user may not know the exact unit they wish to convert to, as they may not know the magnitude of the result, so specifying "metric" will allow the system to choose the most appropriate unit.

7. **The system shall correctly process common abbreviations/pseudonyms for any units that it "knows" about (e.g.** `kg`**,** `kilos` **for kilogrammes;** `microns` **for micrometres).**
Rationale: As part of the natural language usage, it is very unusual for a user to write "kilometres"; "km" is far more common and easy, with no ambiguity.

8. **The system should have a web-based user interface.**
Rationale: This allows the system to be accessed from a wide variety of machines and locations, and means that users of the system are not limited to a single operating system, although this requirement does not force the web server to be runnable anywhere.

9. **The system should allow the user to decide between their preferred unit type—whether they mean an imperial unit or a U.S. unit, for example.**
   Rationale: Due to the web-based user interface requirement, the user domain is increased to potentially world-wide, and different users will have different needs and expectations.

10. **The system may offer the user the option of deciding whether, for example, `500m` is represented as such, or as `0.5km`.**
    Rationale: This adds to the user experience, but the system should use a sensible default.

11. **The system shall not take longer than six months to complete.**
    Rationale: This is the deadline for the project.

12. **The system shall only require one programmer.**
    Rationale: No outside help is allowed.

13. **The system shall be properly documented.**
    Rationale: Proper documentation allows later programmers to understand the system, and modify it if necessary.

14. **The project should result in several new CDs being submitted for approval, via Professor Davenport.**
    Rationale: Part of the project specification, most likely related to unit abbreviations and additional units.

3.4 System Requirements

The system requirements are an expansion of the user requirements, split into *functional*, *non-functional*, and *domain* requirements.

3.4.1 Functional Requirements

This section expands on the user requirements that deal with the system's functionality. From a requirements analysis, it seems that the following user requirements are functional requirements, and so this more detailed list will be based upon these: 1 (parts), 2 (parts), 3, 4, 5 (mostly), 6 (mostly), 7, 9, 10.

1. The system shall correctly convert between compatible units stored in Content Dictionaries (CDs) on the OpenMath website.

   (a) The system shall be able to accept user input regarding which units and the quantity they wish to convert.

i. The system shall notify the user when one or more of the units required for their conversion is unknown

A. The system shall offer the user a chance to upload/link to the file containing the requested definition

(b) The system shall be able to read and process correctly the unit definition Content Dictionaries currently found on the OpenMath website, with little or no user guidance regarding where to find these.

(c) The system shall be able to find and process correctly any new unit definition Content Dictionaries that are added to the OpenMath website, providing the names remain standard (containing "unit" or "dimension").

2. The system shall be able to use further unit definitions, in OpenMath's XML syntax.

(a) The system shall allow the user to upload/link to a new valid Content Dictionary (not to the OpenMath site), which it shall process and use correctly. If an "official" Content Dictionary shares a name with a "contributed" one, the "official" one shall take precedence.

i. This should be possible if either a conversion is not in progress, or the system has determined during a conversion that a unit name was not known.

ii. The system may automatically generate a Content Dictionary for the user based on conversion factors and unit names/abbreviations provided by the user.

(b) The system shall ensure that if any unit names in the new Content Dictionary clash with existing names they are processed correctly.

i. The system should prompt the user to determine which unit should take precedence

A. The system should inform the user of the two definitions, so that the choice is clearer

ii. Failing this, the definition in the "official" CD shall take precedence.

(c) The system should report if the Content Dictionary supplied is invalid.

i. The system shall report if the Content Dictionary supplied is invalid such that one of more of its units cannot be parsed.

3. The system may be able to understand a natural language expression in English for conversion (e.g. 5 miles in km).

(a) The input may contain unit names, abbreviations or pseudonyms of unit names (microns is an abbreviation, tonne is a pseudonym), or unit symbols (e.g. m for metre), or legal SI prefixed versions of any of the above, so these shall be accepted.

(b) Otherwise, the system shall accept the user's input in two separate parts (input quantity and unit, output unit), providing this is clear and simple to use.

4. The system should allow the user to specify which their preferred unit type is, where there is ambiguity.
   (a) The system shall choose one by default, if the user has not/can not specify one.
5. The system shall flag up an error if the user attempts to convert between two incompatible units.
   (a) The system shall explain why the units were incompatible
      i. The system may suggest alternative units that the user may have intended.
6. The system shall be able to convert to a given measurement standard.
   (a) The system shall be able to determine the most appropriate unit(s) for that standard and input.
      i. The unit chosen should have an appropriate SI prefix, if it is possible for that unit to take one, and the addition gives a more meaningful result (`1 metre` is an example of a case where adding a prefix is *not* appropriate, while `0.000001 m` is an example where it would be).
7. The system shall correctly process common abbreviations/pseudonyms for any units that it "knows" about (e.g. `kg`, `kilos` for kilogrammes; `microns` for micrometres).
   (a) This could be performed via one or more `unit_abbreviations` CDs, which we will need to write.
   (b) Unit symbols, including SI prefixes, should also be recognised—although the user is unlikely to type `μ` for `micro`, and therefore the system, if it provides this feature, should allow the use of `u` as well as or in lieu of `μ`, so would recognise `um` to mean micrometres, for example.
8. The system may offer the user the option of deciding whether, for example, `500m` is represented as such, or as `0.5km`.
   (a) The system may offer the user the option of whether the program can override their choice of units with a suitable SI prefix (for example, if the user requests a conversion into `metres`, and the result is `10,000 metres`, if this feature were implemented, an option would be available to instead represent this as `10 kilometres`, or possibly `10 km`.
9. The system may offer suggestions to the user as they enter the desired units, for example, if the user types the first few letters of the prefix "kilo", the system would suggest the prefix itself, and also any units that have that prefix (kilometre, kilogramme, and so on).
10. The system should provide the ability to modify the number of significant figures to return in the result.

3.4.2 Non-functional Requirements

This section expands on those of the above user requirements deemed to be non-functional requirements. From an initial requirements analysis, these seem to be: 1 (parts), 2 (parts), 5 (parts), 6 (parts), 11, 12, 13 and 14.

1. The system shall not take longer than six months to complete.
   (a) The system shall be fully documented and tested in this time
2. The system shall only require one programmer
   (a) This programmer shall also do all of the documentation and testing
3. The system shall be properly documented.
   (a) The documentation shall be written using a standard technique so it can be readily used by anyone with experience of the language
   (b) Additional to the documentation shall be several new Content Dictionaries.
4. The project should result in several new CDs being submitted for approval, via Professor Davenport.
   (a) These CDs should contain unit abbreviations.
   (b) These CDs may contain additional units.
5. The system shall return the result of a conversion involving any two compatible built-in units to the user within 10 seconds of receiving valid input.
   (a) The system shall additionally be able to report problems to the user in this time. These problems may be with the input being invalid, or a problem relating to the system.
6. The system shall be able to convert between two compatible units, where at least one is not built-in, and provide the result to the user within 20 seconds of receiving the initial valid request.
   (a) The mechanism for processing new units and working out how to calculate between them shall not mean that the overall conversion takes more than twice the time of a "standard" conversion.
   (b) This time constraint includes the time taken to determine that the unit is unknown and report this to the user, as well as the time taken to process the uploaded files and return the result. Upload time is not included, as the system has no control over this.

7. There is no particular requirement for hard-disk or memory usage limitations, but the system should not use an unreasonably large amount of either. The front-end in particular shall not use an unduly large quantity of resources, as it should predominantly only be displaying information.
   (a) The specifics of what is "reasonable" in both cases will depend on the language being used. The Content Dictionary files are all text-based, and will generally be reasonably small in size, at most ∼10KiB, so raw storage of these would not be a problem, but the size of the system is not expected to be dominated by these.
   (b) The system design need not go out of its way to minimise disk and memory usage, but reasonable steps should be taken to ensure that neither is being unnecessarily wasted.
8. When a user has supplied a Content Dictionary, the system shall not store it indefinitely
   (a) The system should store the new Content Dictionary for a period of 5–30 minutes, to allow immediate and short-term reuse.
9. Error messages returned by the system shall be clear and appropriate to the error that occurred.
10. When the system is instructed to convert to a particular standard, rather than a particular unit, it shall attempt to determine which, out of the definitions it currently has, are the best units to respond with. This shall not take more than twice as long as a regular conversion between explicit units.
11. The back-end part of the system shall be implemented in a language that is widely supported, preferably Microsoft's C#, due to its speed.
12. The program code shall follow defined standards regarding variable naming conventions, and the like, so that the system can be easily maintained in the future.

3.4.3 Domain Requirements

The domain requirements specify any additional constraints on the system due to the domain the system will be used within.

1. The system shall have a web-based user interface.
   (a) The interface should be usable by as many users as possible, allowing for users with disabilities.
      i. The front-end of the system shall run in a *standard*, *reasonably modern*, web browser, using only standard XHTML 1.0 and, possibly, JavaScript, for non-critical functionality.

ii. The web pages of the system shall fulfil the legal obligation of being accessible.

(b) This effectively makes the system portable, in that it can be run anywhere. However, the back-end may or may not be written in a portable language, but the choice should be an appropriate one.

3.5 Discussion of Requirements

There will now be a brief discussion of some of the requirements which merit this.

Regarding User Requirement (UR) 4, and Functional Requirement (FR) 3, the ability to process natural language input is slightly secondary – it would be just as acceptable for the system to accept a user's request in the form of two inputs – one taking the source quantity and unit, and the other the destination unit or standard, as for it to take a single input which contained both using a reasonable separator ("to", or "in", for example). Both of these forms were observed during the research undertaken for the literature survey, and both seemed intuitive. It is slightly preferable to have the natural-language-based single input, but the main aims of this project lie in the development of the back-end of the system.

UR6, FR6 and Non-Functional Requirement (NFR) 10, relate to the need for the system to be able to convert to appropriate units of a given measurement standard, "Appropriate" has different meanings depending on the standard: With imperial units, for example, it makes sense to convert to several different units, each of smaller magnitude: `2m to imperial` could either return `2.19 yard` or `2 yard 6.74 inch` (values rounded). With metric units, however, it generally does not make sense to convey `6 foot` as `1 m 8 dm 2 cm 8 mm 800 µm`, rather `1.8288 m` is appropriate.

The requirements for specifying preferred unit types (UR9,FR4) has the following reasoning behind it. For example, a typical British user who wishes to convert to/from pints will probably mean "imperial pint" (that is, ∼568ml), while an American user, when specifying to use pints will probably mean one of the U.S pints (liquid or dry measure). Therefore, the system needs to have some method for the user to specify which one they mean, and it preferably should be such that neither the British or U.S. user (in our example) should have to do more to specify their typical choice. This means that there should be a setting that effectively instructs the system to default to a particular unit when there is ambiguity. This is particularly tricky because, although it is usually possible to just specify imperial for a given unit, U.S. units often have several variants, particularly for volume, when there are both dry and liquid measure versions. Another example is that of the "long" or "short" ton.

Regarding the error when the units are incompatible (UR5,FR5,NFR9): as noted in Section 2.4.13, where the converter performs the conversion regardless, this is somewhat confusing, and should be avoided.

It may be non-obvious why the user might need to be able to supply new units when they are not performing a conversion (UR3,FR2(a)i), however, this requirement means that, prior to performing a "to imperial" or similar conversion, the user can provide new units to the system, which otherwise would not be prompted for.

As a consequence of FR 1(c), the system cannot simply have a built-in set of CDs, or even simply periodically update them from the OpenMath website, but must use all "current" ones at all times.

The reader will note that there is no explicit requirement for the system to be able to interoperate with other systems: however, it would be reasonably expected for the system to allow other systems to make use of its service, by not having a complex interface.

Chapter 4

Design

4.1 Introduction

Prior to designing the system, two aspects need to be investigated. One is which areas of a proposed system might meet which requirements, and the other is how parts of the system might be implemented in general. At a fairly early stage of the design, it seemed sensible to attempt to implement prototypes of parts of the system to ensure that the design was feasible. These sub-system prototypes will not be discussed, but the effect they have on any design changes will be.

4.2 Relation to Requirements

As stated in the preceding section, it is necessary to determine which parts of the system are likely to have to fulfil which requirements, and that is the aim of this section. As noted in section 2.12, a design with a back-end which performs most of the processing, and a separate front-end which mostly performs the display seems to be appropriate, and that is the sort of design that will be focused upon. Each system requirement in the preceding chapter is covered in the next three sections, with details of of which part of the system is expected to cater for it.

4.2.1 Functional Requirements

1. Both back-end and front-end will need to facilitate this
2. Both back-end and front-end will need to facilitate this
3. Probably only the front-end will need to facilitate this; before putting it in a machine-readable form for the back-end
4. This would either be only facilitated by the front-end (passing the correct choice to the back-end), or by both
5. Both back-end and front-end will need to facilitate this
6. The back-end will need to facilitate this
7. This would either be only facilitated by the front-end (passing the correct choice to the back-end), or by back-end only
8. This will need to be facilitated by both front-end and back-end
9. This would need to be performed by the front-end only, although the list of suggestions would presumably be supplied somehow by the back-end
10. This could be supported by front-end only, or both, with the front-end passing a user-specified value to the back-end, which it honours.

4.2.2 Non-Functional Requirements

1. This can only be facilitated by a suitably simple design for the whole system
2. This can only be facilitated by a suitably simple design for the whole system
3. This can only be facilitated by a clearly written specification and fully commented code in both subsystems
4. This is outside of the design entirely.
5. Both subsystems will need to be involved in this, although predominantly the back-end, ensuring it performs all processing as quickly as possible
6. Both subsystems will need to be involved in this, although predominantly the back-end, ensuring it performs all processing as quickly as possible; however, the front-end will need to convey more information to the user in this case, and this should performed quickly and effectively.
7. The back-end design should not use more memory than necessary, although if slightly more memory allows a much more efficient design, this is reasonable. For the front-end, no unnecessary media items should be included, and as much of the complex processing as possible should be left to the back-end.
8. This will probably be facilitated by some part of the back-end, although it is possible for the front-end to facilitate it.
9. This will probably mostly need to be facilitated by the front-end, but the back-end will need to ensure it can return errors in a way that the front-end can process simply and effectively
10. This will need to be facilitated by the the back-end.
11. This is clearly only relevant to the back-end of the system, although as part of the language choice it must be considered whether the back-end can be called by the front-end.
12. This will need to be part of both the front-end and the back-end, although variable names will be chosen at implementation stage, to allow the programmer more freedom.

4.2.3 Domain Requirements

1. This requirement clearly can only be met by the front-end design, which will need to be suitably simple in terms of UI design.

4.3 The Designs

With this is mind, a design was put forward. The evolution and detail of the back-end of this design are in Appendix A, along with commentary examining what was changed and why, and Appendix B covers the same for the front-end. In order to make some decisions, it was necessary to consider the language used, and therefore it was decided that this design would be implemented in the latest versions of C# and PHP, as had been anticipated in the Literature Survey and the Requirements chapters, based on the reasons outlined in the Project Proposal, found in Appendix G[1]. The general reason for design changes was that during early implementations[2] it was realised that a particular aspect was either incorrect or caused the implementation to be unnecessarily complicated. The changes made through the preliminary designs to the final design is documented in Appendix A, and the most interesting features of the final design being described in detail in this chapter. The discussion here focuses on how the requirements have been met, and then explains in detail particular parts of the design that merit discussion.

4.4 Back-end Design

This performs the main bulk of the processing. It can be split into several sections—the part which loads the data files in, the data structures used to store the content of those data files, and the processing that is performed on the data structures to obtain the result. This chapter will explain how the various requirements are met by the system, and also explain particular design decisions.

Firstly, a few design decisions that do not reflect requirements will be explained, before examining the back-end design in relation to the requirements. More detailed discussion of the design can be found in Appendix A and chapter 5.

4.4.1 Graph

It was realised in Section 2.6 of the the Literature Survey that the general problem of unit conversion maps very nicely to the concept of a graph, with units as nodes, and edges existing between units for which a conversion was defined. Therefore, this structure was concentrated on.

[1] Although it was not prescribed that the latest versions of either language be used, there seems no reason to use an older version; there are no interoperability requirements that might cause this, and therefore the latest versions will be used

[2] As mentioned in section 4.1 on page 37, early on in the design, implementations would be prototyped, as part of a feasibility study.

4.4.2 Operators

As part of the prototyping phase, an early method attempted to load definitions for every OpenMathSymbol(OMS) it came across. This worked well until it started loading the CD arith1, which in turn required it to load in many other definitions in a huge range of CDs, which were entirely irrelevant to unit conversion. At this point, we realised it was entirely unnecessary. This was overly complicating a system which, really, only needs to use the symbols `eq` in `relation1`, and `times`, `divide`, `plus`, `minus` and `power` from `arith1` (it has been decided in Davenport & Naylor (2003) that these are the sensible choices for units). Therefore, we decided to hard-code a list of known operators, and not load their definitions.

4.4.3 Relation to Requirements

Functional Requirements

1. The parts of this requirement relating to using definitions on the OpenMath website are fulfilled by the mechanism of loading the definitions—it is designed such that both current definitions, and any new ones, will be available to the user, and the system will work if it is unable to connect to the OpenMath server, providing it has done so at least once previously. The compatibility of units is determined by their being on the same graph, and the error codes can be used to inform the user whether the units were incompatible, or one was unknown. The unit loading mechanism additionally looks in a local directory for user-uploaded files, which are also loaded in.

2. The uploading folder covers part of this requirement, and the XML parsing engine covers the XML format part. As the OpenMath website is consulted before the user-uploaded files, this means that the official CDs will take precedence over user-contributed ones. If a unit shares the name of another unit in another CD, this will be treated as a separate unit. However, if a CD has the same name as an official one, the user is notified, and the conversion does not take place. Any errors encountered in parsing the CD are returned to the front-end with a particular error number.

4. Since the requirements were written, it has been realised that OpenMath unit names are inherently geared towards U.K. users[3]. Therefore, this design uses the fact that the names are different for the different units, rather than have the user specify a default. Therefore, the system chooses the default.

5. This is dealt with at the command line parsing stage—when the units have been recognised, if they are incompatible, an error is returned to the front-end of the system.

[3] `pint` in `units_imperial1` refers to the Imperial pint used in the U.K., but the equivalent in `units_us1` additionally specify that they are U.S units: `pint_us_dry` and `pint_us_liquid`, although this is slightly redundant.

6. This requirement is covered by the combination of the unit type being stored in the OpenMathUnit object, and special conversion code that attempts several conversions and determines the best result to return. Prefixed versions of the units are included, where appropriate.

7. Parts of this requirement feature in the design, in that an OpenMathUnit contains an Abbreviations field; however this function needs to wait until the unit_abbreviations CDs are written. See 6.1 for more details.

8. No attempt to cater for this requirement is present in the design. However, where the system has a choice of results such as `500 metre` or `0.5 kilometre`, it will choose the one with the smallest absolute log.

9. The known unit names, including prefixed versions, are written to two files, one for prefixable units and the other for non-prefixable units, as appropriate, for use by the front-end.

10. Although this requirement could be implemented in the back-end, it has been decided to implement it as part of the front-end instead. See Section 4.5.1 for more details.

Non-functional Requirements

5. This requirement cannot be guaranteed by the design, which does not have any timing information available to it. However, aspects of the design are specifically intended to speed up the system, such as downloading the data files to a local directory, and using HashTables instead of arrays in several instances. It will not be known until the implementation is finished how successful these attempts are.

6. This requirement also cannot be guaranteed, but the user file is placed by the front-end in a directory local to the program, so again processing should be rapid.

7. The system is designed such that it does not use more memory, certainly in the data storage classes, than absolutely necessary, so as to remain low on memory usage. In addition, instead of storing OpenMathUnits in most places, an equivalent OpenMathSymbol is stored instead. This is to reduce memory usage, at the cost of taking slightly more time to process, because unit lookup involves an extra step.

8. This requirement is technically not met necessarily, as, rather than ensuring user files are deleted 30 minutes after being uploaded, when the system runs, it will delete those user files that are more than 30 minutes old. This avoids the additional complexity of using a scheduler.

9. The back-end design takes into account what data the error messages displayed by the front-end will need, and supplies this, by returning one of a predefined set of error codes, and printing any additional error information to the `stderr` stream. The back-end design ensures that it does not print any other information to this stream.

10. The design includes specific functions to convert to certain standards, to fulful this requirement.

11. The back-end is being implemented in C#, so this requirement can be fulfilled.

12. Although this requirement is not specified in the design, the implementer will need to meet this requirement.

Domain Requirements

None apply.

4.5 Front-end Design

This contains the user interface, with some processing. The detail can be found in Appendix B.

4.5.1 Significant figures

As discussed in section 5.2 of the paper in Appendix H, it is important to only perform any rounding of the result at the end of the calculation. The precision of the result given should be no greater than any of the numbers used in intermediate calculations. However, it is not possible for the system designed here to know how many significant figures have been used in the calculation, because it cannot tell whether a number it uses is an exact fraction, which would not require rounding, or a calculated value, which would require rounding, and therefore, it is not possible for the back-end of the system to decide for itself how precise a value to return. Then, the options remaining are that the front-end takes input from the user on this and either passes this to the back-end, or the front-end performs the rounding itself, based on the user's choice. In this design, we elected to go for the latter option, mostly because, from looking into features of the languages used, it seemed easier to write a function to manipulate significant figures in PHP than C#.

4.5.2 Relation to Requirements

Functional Requirements

1. The front-end is designed such that it passes the user input to the back-end as an appropriate set of command-line parameters. The front-end also displays the result/error returned by the back-end, with a suitable message to go with it.

2. The front-end has a button to allow the user to upload data files, and it also recognises the output from the back-end that means a unit is not recognised; in either case it

prompts the user in the same way—with boxes for a CD file, the corresponding STS file, and an abbreviations file, although in the error case, it explains why it is necessary and which unit was not recognised.

3. The design went for the two-input approach, with a box for the source quantity and unit, and a second box for the destination quantity and unit.

4. Since the requirements were written, it has been realised that OpenMath unit names are inherently geared towards U.K. users. Therefore, this design uses the fact that the names are different for the different units, rather than have the user specify a default. Therefore, the front-end has no mechanism for choosing a default.

5. The error code received from the back-end of the system is recognised and a message provided to the user explaining that the units were incompatible.

7. The front-end of the system passes the units specified by the user—in whatever form—to the back-end unchanged.

8. No attempt to cater for this requirement is present in the design.

9. The front-end reads the two text files generated by the back-end, and copies the contents into a JavaScript function which offers suggestions, if the user has JavaScript enabled. If support is not available, or the user chooses to turn the suggestions feature off, the rest of the system functions without the suggestions

10. The user interface has a drop-down box to allow the user to specify the number of significant figures that appear in the answer. The front-end then rounds the result returned by the back-end as appropriate.

Non-functional Requirements

5. This requirement cannot be guaranteed by the design, which does not have any timing information available to it. However, aspects of the design are specifically intended to speed up the system, such as the minimalistic design of the front-end, which is strictly text-based. It will not be known until the implementation is finished if these attempts are successful.

6. This requirement also cannot be guaranteed, but the user file is placed in a directory local to the back-end, in the expectation that this will speed up access.

7. The text-based nature of the page is intended to mean the result is given to the user as quickly as possible.

8. This requirement is met effectively by the back-end, so there is no effort made in the front-end design to meet it.

9. The design takes account of the error codes and messages supplied by the back-end, and creates user-friendly output.

11. The front-end is being implemented in PHP, generating XHTML and JavaScript, so this requirement can be fulfilled.

12. Although this requirement is not specified in the design, the implementer will need to meet this requirement.

Domain Requirements

1. The front-end design is that of a web page consisting of XHTML and JavaScript generated by PHP code, and it meets standards defined for usability.

Chapter 5

Detailed Design and Implementation

5.1 Introduction

This chapter discusses in more depth some of the more interesting features of the design, including how they were implemented. It covers various aspects of both the front- and back-ends, and notes problems encountered, explaining how these have been, or could possibly be, fixed. The source code and executables for the front-end and back-end of the system can be found on the included CD, in the units and OpenMathConverter directories respectively.

5.1.1 Overview

This section is to present an overview of how the system will operate, based on the methods defined in Appendix A.3.

1. Front-end: (The user enters the source and destination units, and specifies a number of significant figures).
2. Front-end and Back-end: Validate the user's input. The front-end ensures there are enough input values, while the back-end checks they are in the correct format.
3. Back-end: Download the list of all symbols.
4. Work out, from this, which files need to be downloaded.
5. Download any CD files which are more recent than the current downloaded version, and their STS files as appropriate.
6. Load in dimension files.
7. Load in unit definitions (metric, imperial, US, time + any CD files in <upload directory>). Anything in upload directory created more than 30 minutes before should be deleted.
8. Read STS files and store data in each unit—if any errors, or any units without dimension, return an appropriate error.
9. Create a graph for each dimension, add each unit to the correct graph based on its STS data. Store graphs in a HashTable.
10. For the first unit, look up in the HashTable of graphs for the unit. If it does not exist, see if can generate it as a compound unit, adding any new graphs as necessary.
11. If second unit name is not one of the defined measurement standards, look it up in the same way. Compare the dimensions to ensure they are the same. If one of the units is not found, it is necessary to ask the user for a definition. If find both, but not in the same graph, the system needs to ask for definition with the same dimension

for either of them. The back-end program will exit in this case. The front-end needs to parse the output to determine that additional files are needed for a specific unit name and then get the user to upload both an CD file and an STS file, and possibly a second CD, for abbreviations. Once uploaded, the front-end needs to re-call the back-end program with the same options.

12. If second unit name is "metric", run the "ConvertToMetric" method, as this needs special treatment.
13. If second unit name is one of the other defined measurement standards, call the "ConvertTo" method with that standard as a parameter.
14. Generate Graph edges for the relevant graph—or graphs in the case of compound units.
15. Find the shortest path (or "a path") between the two units using a Conversion object.
16. For each step in this path, work out if have to use the Formal Mathematical Property (FMP) or inverse of the FMP (does the FMP map going from current unit to next unit, or the other way round?). Compute the formula to use. Apply the formulae in order.
17. Return result and error code.
18. Front-end: Read error code, output appropriate message.
19. If no error, round the result to the user-specified number of significant figures.
20. If error requires the user to upload a CD, or if they choose to manually do so, provide file upload boxes for this purpose, and upload the user-specified files to the upload directory, before retrying the conversion.

5.2 Version Control

An important issue to consider before embarking on such a large project is that of version control. A version control system is one that allows the user to store various previous versions of a particular file, and revert to, or compare between, any previous versions of that file. This is useful if for example a modification is made, and then some time later it is found that a previous version worked better, which is a common occurrence in software development. Therefore, it was decided to use the popular source code control system SVN, using the Windows client TortoiseSVN due to familiarity. Frequent check-ins were made to this repository. However, it was generally used more as a backup. Branches were not used, because it was not envisaged that several parts would be under development at once; at least not in a way that would merit the difficulties of merging several branches, so instead, whenever a significant piece or amount of code had been written, the files were checked into the repository. This meant that, although sometimes the files would not compile, or there

were other known issues (which had to be explained in the note attached to the checkin), it meant that if, in the future, any part of the code was found not to have stopped working as expected, it should be possible to find the modifications that caused the change, and revert the relevant parts.

5.3 Implementation Details: Back-end

There are a variety of classes with interesting functionality in the back-end of the system. Those covered here are the OpenMathApplication, OpenMathUnit, Graph, Conversion, OpenMathOperator, and Program classes.

5.3.1 OpenMathApplication (OMA) Class

The most interesting methods in this class are Simplify, Unwrap, and ReverseApplyTo.

Simplify

Since the final design included the OpenMathOperator class and OpenMathNumericType class, this method has had a lot of its complexity removed. However, it serves as a backbone, calling both the Operate method in OpenMathOperator to simplify the OMA down as much as possible, and this class's Unwrap method, to get the result out.

Unwrap

This method operates on the current OMA object, ensures that it is of the form <eq, OMA, number>, or <eq, number, OMA>, and uses the ReverseApplyTo method to apply the number to the inner OMA.

ReverseApplyTo

This operates on the current OMA, which is of the form <operator, OMA, number> or <operator, number, OMA>. It then inverts the operator: (times a b) $\leftrightarrow$ (divide b a), (plus a b) $\leftrightarrow$ (minus b a), (power a b) $\leftrightarrow$ (power a $\frac{1}{b}$). It then uses the assumption that, because of the way the CDs are written, the "b" in all of these cases should be replaced by the OpenMath to apply the OMA to, so it does this, and then calls the OpenMathOperator operate method.

5.3.2 OpenMathUnit

The main method in this class of note is the IsSimple method.

IsSimple

This method attempts to determine whether the unit is a simple one, or a compound one. It is used solely to determine whether a unit can be prefixed, because OpenMath has no way to determine this at present. Initially, it was thought that it could be determined that a unit could take a prefix if it does not have an FMP of its own, or its dimension's FMP is null. This works well for many units, and does not seem to allow prefixing of too many units. However, there are some units that are allowed prefixes but are not caught by this method. This unfortunately meant that several hard-coded cases had to be included, because certain units can take a prefix even if neither of these conditions hold, such as Pascal, Watt, and litre in units_metric1, and bar in units_imperial1. A satisfactory way to encode this need without allowing prefixes for too many units has not been found. This strategy is not extensible, especially for user-supplied units, but it was the only method found to work.

5.3.3 Graph Class

Having investigated the QuickGraph library, it was decided to not to use this and write our own to have the functionality we needed. This class has 3 noteworthy methods. Two, GenerateEdges and RecursiveFinder, work together to add Edges to the graph, and the third, GetShortestPath, attempts to find a route through these Edges.

GenerateEdges with RecursiveFinder

This method examines every unit in the graph. If its FMP is non-null, that is, it has a definition, it uses the assumption that the FMP will be of the form <eq, unitName, formula for getting to other unit>, and therefore only looks in the third part of the FMP. It passes this to RecursiveFinder, which recursively sets anything that is not an OMS or an Operator to null, and return this. GenerateEdges takes this, and determines whether it is a simple OMS or an application. If it is a simple symbol (such as with 1 foot = 0.3048 metre), then it ensures that the symbol is one that is known to the system, and if it is, adds an edge between the two units. If the definition is to an application, and is to a compound unit, then this unit will not be in the same graph, and an edge cannot be created between the two. If the unit is not known, then an exception is thrown reflecting this, so that the front-end can ask the user to supply a definition.

GetShortestPath

After the Literature Survey's investigation of various shortest path algorithms (section 2.6.2, on page 21), it was decided to implement our own Breadth First Search, which will always find the shortest route between two units if it exists. A description of the algorithm follows.

It first ensures that the units are both in the graph. It then uses the method GetEdgesContaining to get an array of the Edges which start or end with the first unit. From now on, when this description talks about "the edges", it means the subset of edges that the first unit is part of. It then looks through all of these edges to see if the other end is the other unit. If one of them is, then this is the only edge required, so a list containing this single edge is returned. If none of the edges fulfil this, then the method takes the "other" unit of each of the edges in turn, and calls GetShortestPath with this unit and the second unit the user specified. In addition, the edge used to pass between these two units (the user's source unit, and the potential second unit) is specified, and this is passed to GetEdgesContaining as an edge not to include in the list. This is to ensure that an infinite loop, going forwards and backwards along the same edge, cannot occur. In this recursive manner, the inner GetShortestPath returns the route for the "rest", so the outer GetShortestPath takes this route, if it is not null, prepends the first edge, and returns the complete list. If it is null, then the outer GetShortestPath tries the next edge in the list, and so on. Eventually, either a route will be found, or it will be determined that there is no route between the two.

This algorithm will now be demonstrated with an example.

Let us say we are trying to convert between mile and mile_us_survey, which will both be in the length graph. Looking at the definitions of the units, it is clear that we should get a route from mile to foot, then foot to metre, then metre to foot_us_survey then foot_us_survey to mile_us_survey. Due to space constraints, in this example, "us_survey" units will simply be referred to as "us". The algorithm does indeed obtain this route, as follows:

- Enter GetShortestPath with source unit = mile and destination unit = mile_us, preceding edge = null
- Check both mile and mile_us are in the graph—they are
- Get a list of edges containing mile, excluding the preceding edge (null in this case). This is a single edge whose other end is foot.
- This edge (mile to foot) does not reach mile_us, so continue
- Go through all the edges in the sublist. This is still only one, (mile, foot).
- For this edge, set the new source unit to be the other end of this—that is, foot.
- Try to get the shortest path between foot and mile_us
  - Enter GetShortestPath with source unit = foot, dest. unit = mile_us, preceding edge = (mile, foot)
  - Check both foot and mile_us are in the graph—they are
  - Get a list of edges containing foot excluding the preceding edge (mile, foot). There are three of these: (yard, foot), (foot, metre), (inch, foot)
  - Examine each of these in turn. As can be seen, none reach mile_us, therefore continue

- Taking the first edge, set the other end (yard) to be the new source.
- Try to find a route between yard and mile_us.
  * Enter GetShortestPath with source unit = yard, dest. unit = mile_us, preceding edge = (yard, foot)
  * Check both yard and mile_us are in the graph—they are
  * Get a list of edges containing yard, excluding the preceding edge. This is a single edge whose other end is furlong.
  * This edge (furlong, yard) does not reach mile_us, so continue
  * Go through all the edges in the sublist. This is still only one, (yard, furlong).
  * For this edge, set the new source unit to be the other end of this—furlong.
  * Try to get the shortest path between furlong and mile_us
    · Enter GetShortestPath with source unit = furlong, dest. unit = mile_us, preceding edge = (furlong, yard)
    · Check both furlong and mile_us are in the graph—they are
    · Get a list of edges containing furlong excluding the preceding edge. This is an empty list, so return null.
  * The returned route was null, so continue onto the next item in the list. However, the list only contained the (yard, furlong) edge, so there are no others to try. Therefore, return null.

  The route from yard to mile_us failed, so try the next edge. This is (foot, metre), so set the new source to be metre, and try to find a route from metre to mile_us.
  * Enter GetShortestPath with source unit = metre, dest. unit = mile_us, preceding edge = (foot, metre)
  * Check both metre and mile_us are in the graph—they are
  * Get a list of edges containing metre, excluding the preceding edge. There are lots of these, as all the prefixed versions of metre are in the graph, with an edge to metre. However, due to space constraints, we will ignore these for now, because they fail in the same way that the other routes have done so far. We will focus on the only other item in the list, the edge (foot_us, metre).
  * This edge (foot_us, metre) does not reach mile_us, so continue
  * Go through all the edges in the sublist. Again, in this walkthrough we will only consider the edge (foot_us, metre).
  * For this edge, set the new source unit to be foot_us.
  * Try to find the shortest path between foot_us and mile_us.
    · Enter GetShortestPath with source unit = foot_us, dest. unit = mile_us, preceding edge = (foot_us, metre)
    · Check both foot_us and mile_us are in the graph—they are
    · Get a list edges containing foot_us excluding the preceding edge. These are the edges (mile_us, foot_us) and (yard_us, foot_us).

            - Take the first edge in this list, (mile_us, foot_us). This edge DOES reach mile_us, so return this edge.
        - The returned route is not null (it is the single edge (mile_us, foot_us)). Therefore add the current edge (foot_us, metre) to the list to return, and then add the returned route. Return the resulting list.
    - The returned route is not null (has two items), so add the current edge (foot, metre) and then the returned route to the list to return, and return it.

- The returned route is not null (has three items), so add the current edge (mile, foot) to route list, and then add the returned list. Return the resulting list.

We now have the list from mile to mile_us, ready to use in the conversion.

5.3.4 Conversion Class

This class has four methods that perform interesting functions. The CreateConversion method, which uses the Replacer method covers two of them, and the other two, which also work together, are the Perform and PerformConversion methods.

CreateConversion, Replacer

The CreateConversion method works in two modes. Firstly, the dimension of the two units is ascertained, and the graph is consulted to attempt to find a route between the two. If a route is found, this route is stored as a list of edges privately in the Conversion object—no other part of the program needs this information. If the two units are of the same dimension, and that dimension is a "basic" dimension[1], then this is all that can be done. If a route is not found, then in the basic case the two units cannot be converted between. In the case where the dimension is not basic (speed, area, and so on), the second mode, then it is not the end of the story if a route is not found. If a route cannot be found, the dimension is split into its constituent parts from its definition, and then each of the units are split into their component parts, and verified to match the corresponding part of the dimension. If they do, a new conversion object is created for each corresponding constituent part. An array of these Conversion objects is stored in the Conversion object, and two OpenMathApplication objects are declared, one specifying how the units were split apart, and another specifying how to put them back together, as depending on the units these may be different. A difficulty with this arises if one unit takes several of the constituent dimensions: if converting between volumes, where the dimension's definition is in terms of length $\times$ length $\times$ length, which is easily mapped to a unit which is defined in terms of three length units (cubic metres, for example), but does not work correctly if one

[1] A basic dimension is one which is not defined in terms of any other dimension: speed is defined in terms of length and time, which in turn are not defined in terms of any others—they are base quantities—so the latter two are both basic dimensions, but the former is not.

unit takes up two or more parts, for example if using litre as a volume, the litre takes up all three of the lengths. This is where the Replacer method comes in. It splits the unit that takes up several parts into individual units, so individual parts can be converted between. The CreateConversion method sets up the decombine and recombine parts. The decombine part is the OMA used to perform any additional calculation on the input before starting: when converting between acres and square metres, for example, as well as needing to be split into square yards, the acre also has a multiplication to do (1 acre = 4840 square yards). This goes into the decombine stage, and the inverse of it occurs before the conversion—in this example, the input is divided by 4840. The recombine stage specifies how to put the intermediate conversions back together to make the whole, typically by multiplying them all together, or dividing the distance part by the time part in the case of a speed conversion.

A difficulty with the CreateConversion method as described here occurs when we convert a compound unit to a prefixed named unit, for example, a force unit divided by an area unit to kilopascals. Then, because kilopascals is defined in terms of Pascals only, the conversion cannot be found, because the dimensions of none of the constituent units (only the whole) of the force over area unit can be converted to a pressure. To avoid this, a special case was added where if the second unit appears to be a prefixed unit (it is impossible to tell programmatically, but a heuristic is used) and the first unit is not prefixed, then a conversion is attempted between the source unit and the non-prefixed version of the destination unit, with an additional step added to then reapply the prefix, by reversing its effect (if converting to kilo-, which is 10^3, the result needs to be 10^{-3} times the size of the unprefixed unit).

Perform, PerformConversion

The Perform method passes the input, initialised as part of the Conversion object's constructor, through the decombine stage if this is not set to null. It then sees if the conversions list has anything in it; if it does, it gives the first item in the list the result of passing the input through the decombine stage, and any other Conversions in the array get given the input value 1. Then Perform is called on each Conversion in turn. Once they are all completed, the recombine part is used to combine the values, by replacing each successive OpenMathSymbol in the recombination with the results of the conversions, and then calling Simplify on this to get a value out.

If the Conversions array is empty, then the method checks to see if the route list of Edges is null. If it is not, PerformConversion is called, on the decombined input, and the result is stored in the Output field of this Conversion object. PerformConversion goes through the list of Edges, determines which way to go through the conversion (that is, whether the current unit is the first unit in the edge or the second. It calls one of the Replace methods in OpenMathApplication to replace the OTHER unit with the current value in the conversion for this edge. It then calls Simplify on the conversion, and updates the currentValue and current unit. It continues to do this until the list of edges for this conversion has been traversed, and then stores the result in Output. Although it may seem counter-intuitive to replace the other unit by the value, meaning that the conversion OMA has both the

current unit name and the current value (which is in place of the other unit name), when the result is unwrapped, this yields the correct result, and was found to be necessary for temperature conversions in particular.

If both Conversions and route are null, after the CreateConversion method has been called, then the conversion cannot be performed.

5.3.5 OpenMathOperator Class

This class creates and maintains singleton copies of the 6 recognised operators, equals, plus, minus, times, divide and power. The most interesting method is the Operate method.

Operate and its helpers

This method takes an OpenMathApplication, and performs its operation, by way of several helper methods: opEquals, opPlus, opMinus, opTimes, opDivide and opPower. Firstly the Operate method simplifies any OMAs within it by calling their Simplify method. If any part of the OMA is a OpenMathSymbol at this point, the method returns, because nothing more can be performed. Otherwise, the appropriate method is called. Each helper method checks that all arguments are numeric, and replaces the OMA with an equivalent numeric value. All the helper methods allow any number of operands apart from power, which is strictly limited to two, the base and exponent; however, any operand can be arbitrarily complex, built up with OMAs. The opEquals method in fact does nothing at all, but it was included for completeness. Once the helper methods have been run, a further simplification process occurs, and the result is returned.

5.3.6 Program Class

There are quite a number of interesting methods in this class. The methods we will examine in this section are ConvertTo, ConvertToMetric, FindFileList, GetDerivedUnit, GetLocalFilename, LookUpUnit, MakeGraphsFromUnits, ReadDefsFromCDs, and ReadUnitsIn. In the following sections, they are grouped together logically, rather than alphabetically. Before examining these methods, however, it is necessary to examine some of the reasoning behind their use, because it was a requirement of the system to be able to use the most up-to-date units, so it is worth examining how this design achieves this.

Data Loading

One of the fundamental problems the design has to cover is the problem of ensuring that at all the times the system is using the most up-to-date definitions available. It also has to be able to work quickly. It is felt that the proposed design fulfils all of these criteria, with the added advantage that if, for whatever reason, the system can no longer access

the OpenMath website, it will still be able to function. In order to load the definitions into the program, several actions need to occur. Firstly, the program needs to determine which files it needs to load. Then, it needs to locate these files and download them, and read the data in. We wanted to make the system as self-sustaining as possible, and therefore have decided that the best mechanism for this is to just provide the system with the URL `http://www.openmath.org/cdindex.html`, which has a list of all the symbols defined in any of the official CDs on the OpenMath website, along with a link to the CD in which the symbol resides. The system will read this file and determine the location of any unit/dimension/physical constant CDs, by looking for identifiers such as "units_metric". Once the list of CDs has been found, the file names can be determined, and they can be downloaded. We realised that this would be quite time consuming to download the files every time, and then do the next part of data processing every time, so an early idea was to have a separate program to download the files and do as much of the data processing as possible in advance, and then load the processed data into the actual conversion program, which would run on demand. However, again, in an early stage of implementing, we realised that it would be far more efficient to download the files if and only if they had been modified since the last time they were downloaded. Since we were using the HyperText Transfer Protocol (HTTP) to download the files[2], we could set the LastModified header to the date/time on the local file, to ensure we only download it if necessary. This vastly improved the performance of the program—the network access is by far the biggest limiting factor, and this cost was significantly reduced, so much so in fact, that it was decided a separate program was not necessary, as loading the files and processing them takes very little time at all. There are lots of advantages to this approach—provided the system can access the OpenMath web site on the first attempt, it will from then on work successfully, and will update its files if and when it can access any new ones. The disadvantage is that the unit CD names need to remain standardised, for our system to be able to recognise them, when on the OpenMath website. The user-uploaded ones do not have to follow this standard necessarily, as will be explained in the discussion of the FindFileList method.

ConvertToMetric

This method examines the source unit to determine what dimension it is, and then retrieves the correct graph from the HashTable of all graphs. It then obtains all of the metric units in that graph using the GetUnitsOf method in the Graph class. If there are none, it returns null. Otherwise, it attempts a conversion from the source quantity/unit to each in turn, and stores all of the results in an array, and the absolute log[3] of each result in a different array of the same size. If a conversion cannot be performed, it is removed from the list. Once all conversions have been attempted, the result whose absolute log is the smallest is returned. This method returns a string, rather than just a floating point number, because it also needs to identify which unit was chosen. The reason for ConvertToMetric being a separate method is that it only returns a single unit, while ConvertTo attempts to find a

[2]This is the most obvious option available, and seemed to be reasonable.

[3]the natural log, although the base used is immaterial

combination of units.

ConvertTo

Firstly, this method checks that the unit type specified is not metric—if it is, it returns the result from ConvertToMetric instead. Otherwise, this method examines the source unit to determine what dimension it is, and then retrieves the correct graph from the HashTable of all graphs. It then obtains all units in that graph of the specified unit type. If there are none of these, it returns null. It attempts to convert to each unit in this list in turn. If a conversion fails it is removed from the list. The output from each conversion is stored in a results array. In addition, the index and value for both the smallest and largest results are stored, although the smallest value is only updated if the value is greater than or equal to 1. Once all the conversions are complete, if the number of results is 0, then the method returns null. Otherwise, the method will try to return the unit which had the smallest result, because this implies that it is the largest unit. However, if the largest result is less than 1, this will be returned instead.

The method then stores the integer part of the smallest result, before doing the reverse conversion to determine how much of the source unit is left, and the source quantity is updated accordingly. It was found that, due to rounding errors, with large numbers the original quantity sometimes ended up being less than the quantity to remove from it—in this case, the new quantity to use is set to zero. A string containing the result is updated with the smallest quantity and unit. The unit that yielded this smallest value is removed from the list, and the method continues with the reduced source value and smaller list of units to convert to. Eventually, there are no units left in the list, or the quantity has been reduced to zero, whereupon the result is returned.

Due to how this method has been written, it can work with any new Unit types added to the OpenMathUnit class, unless they need to be handled in the same way as metric units.

ReadUnitsIn

This method first calls the FindFileList method, passing in the path to the index of all symbols defined in any CD on the OpenMath website. This is `http://www.openmath.org/cdindex.html`. It also passes in the path to the upload directory, which is where user-uploaded files go, and the set of identifiers: “dimensions”, “units_metric”, “units_imperial”, “units_us”, “units_time”, “physical_consts”, and “siprefix”. Once it has the list of all the files for each of these identifiers, which are in the form of a 2D array, it then separates them out into dimensions files, metric units files, etc. For each of these, it call the ReadDefsFromCDs method. From this method it acquires an array of OpenMathCD objects for that type of OpenMathDefinition, which for units will have the dimension data associated with each. Once it has read all of the CDs using this method, it then runs the ReplaceSymbols method, which replaces all OpenMathSymbol objects which represent known operators with their OpenMathOperator equivalent. It then calls MakeGraphsFromUnits to generate

the graphs, and calls WriteNamesToFile, for use by the front-end of the system. It then returns the HashTable of graphs.

As can be seen, the decision of what type the unit is is based on the file name, so files which match the identifier units_metric are deemed to be metric units for example. This is because, in OpenMath there is no way to determine the standard a unit belongs to; for this reason, this method attempts to base it on the file name—with a small hack required in FindFileList to ensure if it cannot be assigned to one identifier, it is placed in the imperial file list.

FindFileList

This method takes a URL, a path to a directory and a list of identifiers. This URL is that of the `cdindex.html` page on the OpenMath website, which is what it uses to determine which files to download. The directory is the local directory where user-supplied files get stored. The identifiers are for the different types of files to be loaded, such as *dimensions* or *units_metric*. The method downloads the URL (using GetLocalFilename) and reads it a line at a time. If any of the identifiers in the list are found on a line, it will be part of a link to one of the files—for example, a typical example line might read

```
<td><a href="./cd/units_metric1.xhtml#metre">metre</a></td>
<td>units_metric1</td><td>
```

which would match the units_metric identifier.

This method pulls out the part in quotes, and passes it to ConvertToAbsolutePath, which converts something of the form "./cd/units_metric1.xhtml#metre" to something of the form "http://www.openmath.org/cd/units_metric1.ocd", which is the location of the CD file. This is then added to a 2D array of all the files associated with each identifier, and once the whole file has been read, the directory is also read. Firstly any files that are more than 30 minutes old are deleted. The directory should contain CD files and STS files, but the names may not be necessarily be standardised to include the correct identifier. Therefore, the method attempts to look for one of the identifiers in each CD file name, but if it cannot find it, it assigns the CD to the imperial list instead—if it has been given an identifier containing imperial. Then this 2D array is returned.

This method uses a simple process to effectively fulfil requirement NF 8(a). Rather than ensuring that the file is deleted exactly 30 minutes after upload, using some form of scheduling system, it was decided that the system should instead delete any user-supplied files that are more than 30 minutes old before using any of the files from the directory. This means that it technically does not meet the requirement, but it was felt that this was sufficient justification. A slightly different wording to the requirement, with a subtle change in meaning would be "users shall only be able to use the uploaded file for a period of between 5 and 30 minutes prior to its deletion", and the design passes this.

ReadDefsFromCDs

This method takes a list of paths to CDs, the type of definition stored within those CDs and the type of unit (None if not a unit), a Boolean determining whether to read the STS files (this is currently only used for units), the list of known dimensions, which again is only used for units, an array of CDs to write to, and an array of definitions to write to that have been found for that type. It uses the ReadCD method to acquire the OpenMathCD object represented by each file in the list, then adds the STS information using ReadSTS if the flag is set to true. It outputs an array of CD objects and also an array of all of the definitions from all of those CD objects, for ease of access.

GetLocalFilename

This method takes a file name, and returns a local version of the file. If the file name specified is already local, it just returns this. If the file name is remote, it checks to see if there is a local version in the dataFiles directory. If there is, the last modified date is taken and used as part of an HTTP request for the remote file. The method then attempts to download the file using this last modified date as part of the HTTP header. If it can access it, and the LastModified date is more recent than a local copy, then the file is downloaded, otherwise, a NotModified exception is thrown by the HTTP request, and this is handled such that the local version of the file is used. The method then returns the path to the local version, which in all cases will now be the latest version available to the program. If the remote version cannot be accessed and there is no local version, the method returns null.

LookUpUnit

This method takes a string representing a unit name. It then looks at each graph in the HashTable in turn, and calls the Contains method to determine whether that graph contains the unit. It returns the first unit it finds that matches the name. If it fails to find the unit in any of the graphs, it attempts to create it using GetDerivedUnit, and if this fails it returns null.

GetDerivedUnit

This method takes a unit name and attempts to construct a unit to match it. It does this by attempting to split the name according to various identifiers in the hope that the parts of the unit remaining will be present in one of the graphs. The identifiers it looks for (in order) are "_per_", "_sqrd", "_cubed", and "_", where the latter is found where two units are multiplied together. If it finds one of these identifiers, it then attempts to look up the units. At present, the method only works with two units, but either can be compound units themselves, for example mile_per_hour or Newton_per_metre_sqrd. It calls LookUpUnit on

the two parts it has found, and if both are non-null, attempts to work out the dimension of the resultant unit by applying the FMPs of the dimensions of the two units together correctly—if it has found mile_per_hour, the FMP for the length dimension will be divided by the FMP for the time dimension, for example. Then the list of graphs is consulted to see if the compound FMP matches that of any of the graph dimensions. If it does, the new unit is created and added to the graph. If it does not, a new graph is created for the dimension, and the unit added. This is so that, if it is the source unit, as long as the destination unit has the same dimension the units can be converted between, even if the system does not recognise the dimension. If the unit has been created successfully, it is returned, otherwise null is returned. The method does not know whether the unit is the source or destination, so if a dimension is added for the destination unit, then clearly it is of no use, because the source dimension must be different. However, the program will exit after establishing this, so the memory wastage is minimal.

MakeGraphsFromUnits

This method takes an array of known dimensions, known prefixes, and all known units and creates a HashTable of graphs, one for each dimension, containing all of the units known for that dimension, including prefixed versions as appropriate. The units passed in are in a 2D array, with the first subscript corresponding to the subscripts of the different identifiers used to find them, such as "units_metric", or "units_imperial". The units have by now been assigned an appropriate UnitType, such as "metric". Each graph is created, and for each unit in the list, if it is the same dimension, it is added to the graph. Once this has been completed, the method then proceeds to add prefixed versions of the units which it deems are prefixable. Unfortunately, due to a limitation in OpenMath's unit handling, there is no built-in method to determine whether a unit is prefixable, so the method has to use the following heuristic: if the unit is a "simple" unit, it is deemed to be prefixable. The MakeGraphsFromUnits method calls the GetSimpleUnitsOf method, in the Graph class, first passing in the metric type, and then the imperial type, and the GetSimpleUnitsOf method in turn calls IsSimple to determine which can be prefixed. With the returned units, it calls AddAllPrefixedVersionsOf, passing in the list of prefixes and each simple unit name in turn. This method adds all the prefixed units to the graph, and then regenerates the edges, because each prefixed unit will now need an edge to the unprefixed unit. Once this has completed, the HashTable of graphs is returned.

5.3.7 Generating Compound Units

In some of the existing CDs there are several compound units. These were named by separating the parts of the name with underscores, for example metre_sqrd or miles_per_hr. As has briefly been mentioned, in keeping with this, our system also built up unit names in this way, and they were stored in the graph structure in the back-end in this manner. In the end, all of the compound units were removed from the CDs, because it was felt they were not necessary, as the system can build them up using the dimensions and STS data,

but the internal storage mechanism of using _ to separate the names has remained. This could be altered, and probably will be in future.

Compound units where the two units are divided are straightforward to generate (assuming the definition of one does not involve another to a negative power), because they can be split up and the top half processed separately from the bottom half. When the two units are multiplied together, (e.g. litres to acre foot), this is more difficult because the system has to try to convert an area × length to length×length×length, and the manipulations involved turned out to be too difficult to complete perfectly in the time available. Some kinds of multiplied units are not handled correctly, and this is discussed in section 5.8.

5.4 Implementation Details: Front-end

It was decided that the front-end would be an entirely self-contained web page—that is, the page for entering a query and viewing the result would be the same—so that there was no code repetition, and users can do all the main functions from the same place. This meant that if they chose to upload a file, but then decided to do a conversion instead, this was easily possible for them. Similarly, if they performed one conversion they could easily modify their query to perform a new conversion, even if the previous conversion had resulted in them being asked to upload a file; as it may just have been the case that they accidentally typed a unit name incorrectly. Another advantage was that a user does not need to go to a different page depending on which units they wish to convert, as is the case with some converters—however, this is in keeping with some other converters which allow the user to type their unit choices, rather than selecting them from a list.

Due to the simplicity of the layout designed, it was decided that the PHP code would write all of the XHTML dynamically each time, as there was very little. However, it was decided to separate out as much of the JavaScript functionality as possible into a separate, prewritten, file, functions.js, and for the small amount of CSS that was used, to place this in a separate file, styles.css.

5.4.1 Functionality

As the system used JavaScript for some parts of its functionality, but it is not guaranteed that JavaScript is enabled, in order to not confuse the user, or clutter the display with unusable controls, it was decided that if JavaScript was disabled, such controls would not be displayed. This is implemented by writing JavaScript functions that add the controls to the web page—if JavaScript is not enabled, the controls are not added. Therefore, they only appear when they are usable. An example of such a control was the “Reverse this conversion” link, which will be described later on this this section.

We will now examine in more detail some of the more interesting functionality.

Significant Figures

The method to round the result to a certain number of significant figures is written in PHP, as JavaScript has poor support for it. The function used was found on the php.net website, with a reference in the code. However, it was found that the function only worked correctly with positive numbers, so to resolve this, the function was modified to store the value, but perform the processing on the absolute value, and then afterwards, if the absolute value was not the same as the input value (i.e. the input value was negative), the rounded absolute value was then negated before being returned. The initial value of the significant figures drop down box is 15, which is the maximum available, as this is the maximum precision offered by C#'s `double` data type, and thus by the back-end of the system.

Suggestions

To load the suggestions into the page, in PHP the suggestions files are loaded, and the contents written into JavaScript arrays for prefixable and unprefixable units, and the prefixes themselves. There are then several JavaScript functions used to offer the suggestions. When the focus is on either of the two input boxes, as keys are pressed, the parser function is called. Firstly, if the user has pressed the down cursor key, the first item in the list of suggestions, if there are any, is given the focus. Otherwise, the text box receives the input as normal, and the last word in the current text box is examined to see if it matches any of the units in the prefixable, unprefixable, or prefixes lists. If there are any suggestions which start with what the user has typed, but are not equal (that is, as the user completes the unit name it disappears from the list), they are added to a list of suggestions, and these suggestions are added as links, which when clicked or selected by the user, call the add function, which adds the specified string in place of the last word in the specified text box. As the design suggested that the user should be able to navigate the suggestions with the keyboard, a function was written to allow use of the up/down cursor keys to navigate the list, with the <Enter> key then being used to select the suggestion. The suggestions do not get limited by anything else the user has typed in (for example, not suggesting gramme for the destination when the source unit is metre), because ensuring the dimensions were correct for compound units, when many units could be put together where individually they would not be allowed, would be far too complicated, and, as the back-end of the system has to do this anyway, and it is far easier to do it there, it was decided it was not worth the effort. In addition, it was felt that restricting the suggestions in this way might make the user feel they were making a mistake, when in fact they were not, and the user is assumed to know what they want to do.

Reverse Conversion

As a change to the design, a link was added to "Reverse this Conversion", which, when clicked, switches the units round between source and destination, but leaves any quantity

in the source box as is. This was originally put in as a temporary aid to testing, but it was decided that it was quite useful, so remained.

Calling the back-end

As has been mentioned, the back-end uses underscores to separate compound units. This is not particularly user-friendly, and therefore, in another change to the design, it was decided to allow the user to type compound units separated by spaces instead, as it is not too difficult to convert this to a form suitable for the back-end. This means that, if the user were to type "10 mile per hour", it will become "10 mile_per_hour", before being passed to the back-end. The user input is split into source quantity and source unit from the source box, and destination unit from the destination box. These are then converted to the underscored version, and passed to the back-end of the system by following its interface specification. If the user has not specified a source quantity, "1" is used.

5.4.2 User Interface

Following on from the explanations of what the front-end consists of, it is reasonable to present the final user interface. Figure 5.1 shows a unit conversion where the source unit—"micron"—was not recognised. In addition, it shows the suggestions list as the user is typing into the destination box.

Figure 5.2 shows the same conversion with the source unit corrected to "micrometre".

Finally, Figure 5.3 shows how the system appears when JavaScript is disabled. Notice how the unavailable options no longer appear.

5.5 Decisions and Reasons

There are a few further issues worthy of discussion here.

5.5.1 Physical Constants

We were planning to make use of the physical constants CD, so that users could specify, say, "the speed of light", to be expressed in a different unit. However, the definitions for many of the constants in there were in terms of ranges, rather than a single value, and we could not decide an appropriate way to utilise these—as explained in Appendix H, it is not appropriate to store the range. In addition, other units have 2 Formal Mathematical Properties (FMPs), and we have not reached a firm conclusion on how to use two of these—as explained in Section 7.2 of Appendix H. These multiple FMPs were also a feature in the units_time1 CD, which meant that conversions could not be performed between day and calendar_year,

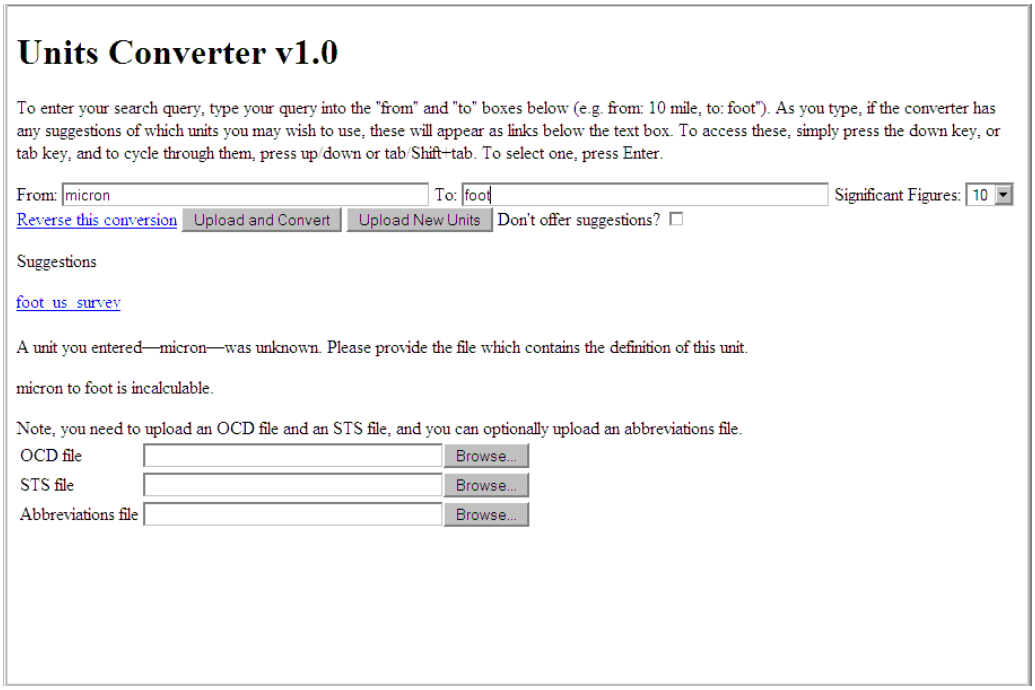


Figure 5.1: The User Interface presented when a unit is unknown

5.5.2 CD Generation

One of the requirements (Functional Requirement 2(a)ii) suggested the possibility of generating CDs automatically from user-supplied content. However, since the requirements were written, a system has been made known to us (Lange 2008) which plans to become a fully-fledged OpenMath CD editor in a future release, and therefore it simply was not practical to consider implementing this feature.

5.5.3 Efficiency

Because the program will not be running for a particularly long time, it was decided that it was not necessary to be especially vigilant regarding freeing up all memory used during the program. This is because it will all be cleared by the runtime environment when the program terminates. C# has a garbage collector, which helps to some extent even when the system is running.

Data efficiency is achieved in some areas quite well. An early idea was to go through all of the length units in the graph, and then add a squared version to the area graph, and a cubed version to the volume graph units. Then, if the user typed a unit in that had a prefix, to prefix these. This led to illegal units such as deci(metre_cubed) so was completely rewritten.

Figure 5.2: The User Interface presented when the input is corrected

Prefixes were added to all units which were allowed to take a prefix, and then added to the area and volume graphs. However, it was then realised that the STS information, such as that speed is a length unit divided by a time unit, was not really being used, apart from in the dimensions. This meant that the solution being implemented was not extensible to the other dimensions. It was realised that it was unnecessary to add modified versions of the length units to the area and volume graphs, because units could just be composed together using the FMP of the dimension. This resulted in a system which only creates and adds such compound units to the graph based on the user's input, and then splits it up to do the conversion between the base quantities as necessary. This achieves a certain amount of data efficiency, because it means that many units are only generated and stored if they are being used. However, one efficiency decision did cause a few problems. It was decided that instead of storing OpenMathUnits whenever a unit was referred to, the system would store an OpenMathUnit for the definition in the graph, but for all other references would leave the units as an OpenMathSymbol. This saved memory, but unfortunately meant the system ran more slowly, especially for unit lookup, which, had OpenMathUnits been stored, the dimension and other properties could be accessed immediately, while with the method chosen the unit object had to be looked up before being retrieved. It would be quite a big change to modify the system to use OpenMathUnits in all the different places, which is why it has not been undertaken so far.

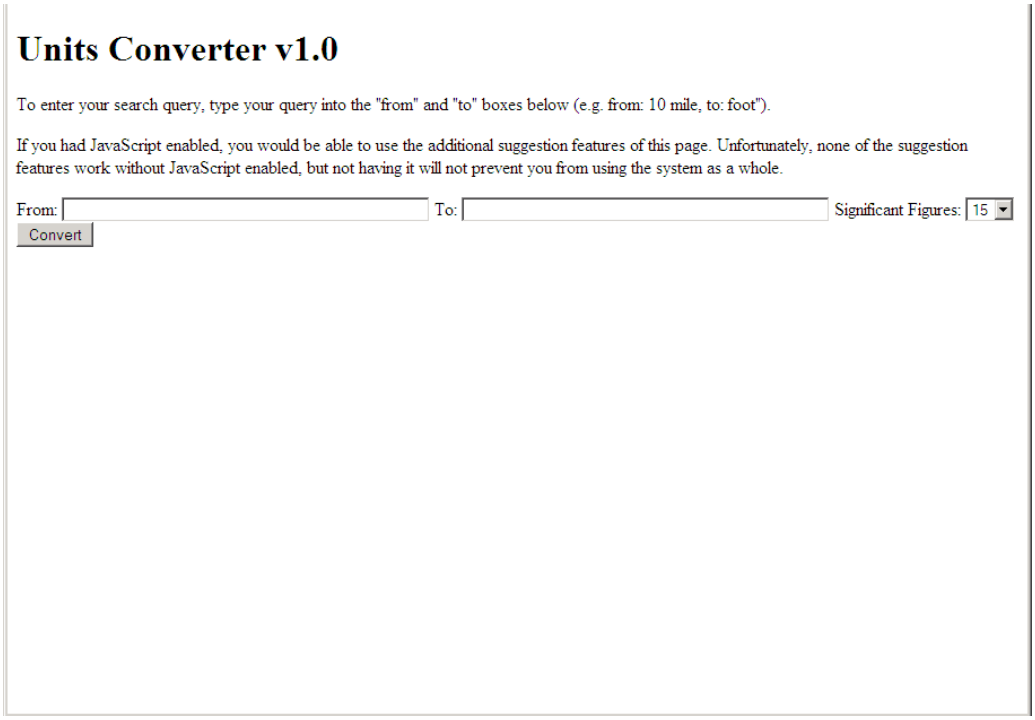


Figure 5.3: The User Interface presented when JavaScript is disabled

5.6 Coding Standards

The standards specified in Appendix D were followed throughout.

5.7 Changes to design

5.7.1 Back-end

There were no real changes to the design of the back-end of the system, after its evolutionary process during prototyping, described in Appendix A. However, as the system allows units to have the same name if from different CDs (see section 5.8, the DuplicateUnit error code is redundant, because it is covered by the DuplicateCD error code. However, it was felt that the code should not be removed because this would require all the others to change, which is bad for the public interface. Also, as will be seen, it is not certain whether the DuplicateUnit error code will be needed in the future, depending on whether unit names should be allowed in multiple CDs. Additionally, the UnknownDimension error code is unnecessary, as any unknown dimension is generated.

5.7.2 Front-end

As a slight change to the design, the addition of a "Reverse this conversion" link appears next to the submit button. In addition, when the system is in the "unknown unit" state, the submit button's text changes from "Convert" to "Upload and Convert".

The final UI design of the front-end was very similar to the initial design, and retained much of its simplicity. However, it had the addition of allowing the user to use spaces instead of underscores to separate compound units.

When a user-uploaded file contains an error which means conversions cannot occur, the front-end attempts to delete this file so users can continue with their conversions. It only attempts to delete user-uploaded files.

5.8 Problems

During implementation, the CDs specified in (Davenport & Naylor 2003) were used. However problems, initially with temperature, and later on with others, were discovered. This led to the changes examined in Chapter 8.

During the course of the project, a great many problems were encountered. Many of them have already been documented in this report, along with the solutions we found. In this section, we will briefly discuss the main problems known that have not been mentioned thus far, with details of why, despite our efforts, the problems are still present.

There is a problem that our program has with the Replacer method in the Conversion class in its present form. Some units need splitting apart to ensure the conversion can progress, while others need to be combined. With the current set up, converting between foot_cubed and pint, for example, works correctly, but the system cannot work out conversions to/from acre-feet, because when it splits the acre into two length units, it has no way to retain the factor 4840. Changing the Replacer method to combine the other unit rather than split fixes this problem, but then converting cubic length units to pints fails, because it gets into an infinite loop attempting to combine the three length units into a volume, when they already are one. It seems more intuitive to combine the units rather than split them, because by splitting up acre, it is not clear where the 4840 should go[4] but because more problems seem to arise from doing this than not, and it was felt a user is more likely to want to convert to pints, two actions were taken. One is that the Replacer method was left as a splitting method, so that more common conversions work. The other was that a definition of acre-foot in terms of cubic foot was added to units_imperial1, so that the system can also correctly convert this without having difficulties with the factor of 4840, because it is present in a usable form in the definition. This should not be necessary in the long term, but is currently needed for any units which are made up of several units, where

[4]This unfortunately means at present it disappears, and the result obtained when the system generates the acre-foot unit is out by a factor of 4840.

one of these units has a multiplicative factor as well as a combination of units. This will be further discussed as part of the further work section of the conclusion.

There are also several problems that do not seem to be entirely with our system, but either relating to use of OpenMath, or unit conversion in general. The first is at least partly a limitation of our system—if two units have the same name but are on different *graphs*, as well as being in different CDs, there is no way the program knows which to choose based on this—if the user has specified to convert from unitA, a length unit, to unitB, which could be either a length unit or a mass unit, the lookup method does not know that the first unit is a length unit and therefore that it should pick the length one rather than the mass one. Therefore it picks one, and there is no way to tell it to pick the other if this is the wrong one. It is hard to say whether this problem is purely a problem with our system, or with the OpenMath data files. It needs further investigation. The problem here is that, because second appears in two CDs, once in units_metric1 and once in units_time1, the system cannot load both of these in if it restricts a particular unit name to only being in one CD. Initially, this is what our system did, in that it stored the units in a HashTable, with the key being the unit name. However, when second was found to appear twice, causing the HashTable adding code to fail with a duplicate key, it was realised that the name was insufficient, so the key was changed to be "unitName-unitCD". This meant that the system worked for second, but meant that it would be impossible to prevent the user adding a unit name that was present in a different CD, and thus the same unit name can be on two different graphs. The problem with our system could be resolved by letting the LookUpUnit method return all the matching units, and then the calling method can decide which to use. However, in many cases, this will just be moving the problem away, without actually solving it, because if, as is the case with second, the two units are on the same graph, it is not clear which one should be chosen. Currently, for second, the system returns the one whose FMP is not null, but in more general terms, if more than one unit fulfils this criterion, the first one it found is returned. This code is experimental in the sense that it has not been possible to check what happens with two units having the same name on the same graph, when both have a non-null FMP. It is unclear whether the unit second should be removed from either of its two CDs, because it is both a metric unit and a time unit, which in turn might have quite a big effect on the system, as several decisions were made—at least in part—because it was in both.

As a subtly different problem to one previously mentioned, the "to measurement standard" conversions only work for base types. This is because at present they have not been written to cater for combining units from several graphs, and picking the most appropriate units for this task. This appears to be a very difficult problem in itself, but it has also been realised that this problem is compounded by time units—if the system were asked to convert to an "imperial speed", what unit should it choose for the denominator? With metric, this is simplified (at present), because there is a second declared to be metric. As can be seen, this problem is closely linked to both the problem of determining what measurement standard a unit belongs to, and whether "second" should appear in both time and metric CDs.

Chapter 6

Abbreviations

6.1 Introduction

One of the areas the project was intended to investigate was the area of abbreviations. There are several parts to this, including unit symbols (m meaning metre, for example), pseudonyms (fermi, now obsolete, being equal to femtometre, tonne being equal to megagramme), and abbreviations, such as micron meaning micrometre.

We started investigating this area, and found that it was far more complex than we had imagined. OpenMath has a prefix system in several CDs, and it was envisaged that a modified version of this would be used to connect abbreviated prefixes ("k-" for kilo-) only to symbols, but not to unit names. That is to say, km would be allowed[1], but kilom and kmetre would not.

In our paper, presented in Appendix H, we examine this area in more detail, and had to draw the conclusion that none of the possible solutions we have come up with are perfect, and therefore for this project, the area had to be abandoned.

[1] and, of course, kilometre would remain allowed

Chapter 7

Testing

7.1 Introduction

Although the requirements were specified, so a testing plan could be developed, there were several open-ended areas, where it was not certain what the implementation would provide. This was because the project had become an investigation into the extent to which OpenMath can be used for unit conversion, and meant that additional tests were added later, as new features were implemented. It also meant that, once it was established that abbreviations were not going to be feasible, tests for this were ruled out. Due to time constraints, although informally each part of the system was subjected to a variety of tests as it was written, formal testing could only cover the back-end as a whole, the front-end as a whole, and the combination of the two as the complete system. Because of the nature of the front-end of the system, many of the front-end tests were in fact system tests, as we chose to ensure that the back-end of the system worked as expected, and then use it during the front-end tests, rather than writing stubs.

It is perhaps worth noting that, while it was helpful to know how well the system performed in relation to these tests, as it was a piece of software that had been developed, it was not the main aim of the project for the software to be perfect; rather, we wanted to find out more about using OpenMath in this area, and in fact the tests themselves reveal little about using OpenMath files for the unit semantics, except that for the most part it seems to work.

7.2 Test Plan

There are two main types of tests covered by the final test plan. Validation testing is intended to show that the software meets its requirements, while defect testing is intended to reveal problems. We wanted to show that the software had the features expected in order for it to fulfil the requirements, but also expose any defects. The main requirement was that it was able to convert units, and it is impossible to test *all* units, so a reasonable subset of tests had to be chosen. As can be seen from the test plan in Appendix E, tests were attempted against a large range of units, but at the same time other aspects of the system were covered by these same tests. The main aim was to test each requirement as thoroughly as possible, as well as investigate the effectiveness of other aspects of the system which, although not in the requirements, had been specified in the design, or even, in some cases, just added during implementation. Although it is bad practice to have parts of the design which are not in the requirements, it was felt that this system was more an investigation into what could be achieved, and as such could go beyond the requirements. The test plan therefore not only needed at least one test per requirement, but also at least one for each additional feature. Although the validation testing was supposed to show that it worked, and the defect testing was hoping to find problems, the validation testing actually revealed some problems that could not be fixed, even though the input data were reasonable.

As each component (down to the level of methods) was written, it was subjected to a few tests created by the programmer to ensure that it worked as expected on its own. Prior to the formal testing of the back-end and front-end, smaller components of both were subjected to fairly extensive informal testing during development, both individually and integrated to varying extents, in an attempt to find and then eliminate as many bugs as possible. This incidentally took the form of regression testing, because it was found that several types of conversion often seemed to fail, notably temperature ones, so every time these were fixed and then new changes were made, these tests were run again to ensure that they still worked correctly. However, due to time constraints, and the size of the system, the main testing came in the form of testing the two main components, and then system testing.

Appendix E details all the tests created, with reasons for the more obscure ones. Our testing strategy was a hybrid of white and black box tests. The main focus was on black box testing; however, as the internal structure of the system was known, it was also possible to write white box tests—both which were expected to pass and fail, to demonstrate the effectiveness of the system. In actual fact, at some points, we did write tests which we were confident would pass, and the tests failed, so we then had to fix the program. The results of our testing is found in Appendix F.

7.3 Test Results Analysis

7.3.1 Back-end Testing: Main Tests

This testing demonstrates the wide range of conversions that are possible with the system. However, it also highlights some drawbacks. The causes of these failures are all either known to us, or we can make a reasoned guess; however perfect solutions to them are not known in all cases. We will now examine the failures, and attempt to explain why they occurred.

It was found that sometimes, test 19 failed because the system substituted pole, as the test case expected, for perch, which is a pseudonym for the same unit.

Test 98, which was converting 10 decimetres cubed into litres, should clearly give 10 as the result. The failure of this test appears to be due to rounding errors, due to the use of finite-precision floating point arithmetic.

Tests 140, 141 and 144 are conversion to measurement standards. However, because the algorithm used only looks in the same graph for units of that standard, it does not work for derived units. There were two reasons for this design decision, and they are explained in the implementation chapter, section 5.8.

Tests 151 and 152 reveal a deficiency in the algorithm for determining the dimension. For compound units, it attempts to find two units divided, a squared unit, then a cubed unit, and finally two units multiplied together. An early version of the system returned the

unknown dimension error when it managed to find one of the above combinations of units, but the result did not match a dimension known to the system. It was realised that this was very restrictive, and instead if a dimension that was not known to the system (by this we mean that the system did not have a definition for that dimension, for example from the Content Dictionary `dimensions1`), the dimension would be inferred, the graph created and added to the HashTable. It was realised that these could not be compared by the names; if the system has "invented" a dimension, it will not know the common name for it, so instead the FMPs of the dimensions are compared at this stage. However, this system is not perfect either, because the units end up grouped together differently, so it might get (mass_length)_per_(time_sqrd), which is in fact the same as force, but the algorithm cannot detect this, because it has created a new dimension mass×length during the process, and this is not part of its definition of the force dimension. Although, as test 157 shows, it can create a dimension which in fact is the same, and because it has found the same "new" dimension for both source and destination, the conversion works.

Test 167 fails due to a problem with the CreateConversion method, where, because the destination unit is prefixed, when the constituent units are obtained, this only consists of the prefixed version of the unit, and the unit itself; both of which share the dimension of the source unit as a whole. The algorithm has some code to avoid this in many cases (as described in Section 5.3.4); however, it is not successful in the case of bar, because when not prefixed, it is still only in terms of one other unit. Attempts to use the same fix as for other prefixed destination units failed. This will need further investigation.

Tests 174 and 175 also reveal a similar deficiency in the dimension-determining algorithm, in that it cannot tell that Watt×second is in fact Joule, or rather, that power×time is the same as Energy. The algorithm will need rewriting to treat dimensions more algebraically, rather than just joining them together blindly.

Tests 182 and 183 demonstrate the same fault as test 167.

Many of the mass conversions reveal a problem with the system's prefixing routine, in that "tonne" should not take prefixes. One source seems to state that "kilo-", "mega-", and so on, are becoming recognised ((Belgian Ministry of Economic Affairs 1970) refers explicitly to the mass form of tonne, as opposed to the "megatonne of TNT" energy unit), but it seems agreed that tonne still cannot take the prefix "milli". This was discovered after the test plan was completed.[1] Despite this oversight, the tests have proved useful in that they demonstrate a problem with the system.

Besides the main tests of the back-end, there were also other tests performed on the back-end, and on the front-end also. As can be seen from the time column of the results table, conversion is well within the boundaries specified. In fact, when the local files were deleted, the time taken to download them and return the result for the first test was also under 10 seconds, and subsequent calls to the program used the downloaded files and therefore

[1] This paper was found through the International Bureau of Weights and Measures, which lends credence to it. Other sources appear to disagree, and the BIPM source itself does not make an explicit statement either way.

ran in a comparable time to prior to the deletion. So it turned out that the 10/20 second limit on getting a result back was very high—the result never took more than 4 seconds to be supplied. However, it was noted that the limiting factor tended to be the network connection—when this was removed, and the system had no choice but to use its local files, the results were much more rapid. So even though no downloading was actually done with the network connection (just last modified dates compared), this was a limiting factor.

7.4 Back-End: Other Tests

7.4.1 Requirements

It was found that, even when the data files were all deleted, the conversion took place in under 10 seconds.

As can be seen, the back-end passes all of its requirements apart from FR 2(b)i, which has been thoroughly documented as a design decision based on second appearing in two Content Dictionaries.

7.5 Front-End

All the error messages, apart from ConversionFailed, DuplicateUnit and UnknownDimension were generated; the last two can no longer occur, and for the former we have not found a conversion that causes this—it requires the program to find a route but fail traversing it.

It was found that the significant figures control has no effect on results in unit standards. This was expected as it has not been decided what to do in this situation.

The bug found in the front-end, although potentially irritating, is not a huge problem, as all functionality still works, it just means that one method of accessing a small part of this functionality is not available.

7.5.1 Usability Guidelines

As can be seen, it fits most of the accessibility guidelines, although there are a few issues with dynamic content which is not accessible, and the issue of being unable to upload files if JavaScript is disabled. These should be rectified, especially the latter.

7.5.2 Requirements

It was found that the front-end of the system met all the relevant requirements.

7.6 System

Most of the System Testing results had already been seen with the Front-end testing, and nothing new was found.

7.6.1 Requirements

It was found that overall the system met most of its requirements—it did not prevent the user from supplying a CD which contained the same unit name as another, but the reasoning behind this has been discussed at length in Section 5.8. The system also did not offer such choices as between displaying 0.5 km and 500 m, but this was only a minor part of the functionality. The system also made no attempt to override the user's unit choice with a prefix, as it was felt that they could use the standards conversions for this. The system was found to pass the rest of its requirements, or they were found to no longer be applicable.

Chapter 8

Content Dictionary Corrections

The Content Dictionaries were found to have several errors and omissions during the course of the project. These were corrected, and the resultant CDs are found in Appendix C. To demonstrate the program working with an entirely new CD, units_beer1 was created, which contains the various sizes of beer container: firkin, kilderkin, barrel and hogshead. As can be seen from the testing, the system could convert between these and the other volume units correctly. Additionally, two CDs of obsolete units were created, and these were also used during testing.

8.1 Temperature

The initial problems noticed with the Content Dictionaries were in the temperature conversions. There were two issues noticed. One was that the conversion for Fahrenheit was incorrect—instead of being specified as $C = \frac{5}{9} \times (F - 32)$, as would have been correct, it was specified as $C = 0.5556 \times F - 32$, which was inaccurate. However, during implementation it occurred to us that this was also incorrect in a deeper way. We realised that there are in fact two kinds of temperature conversion; based conversion and relative conversion. For example, 10°C is $10 \times \frac{9}{5} + 32 = 50$°F, but an *increase* of 10°C is an increase of $10 \times \frac{9}{5} = 18$°F. This meant that as well as correcting the 0.5556, we needed to add *new* temperature definitions for relative_temperature. This led to us considering which other dimensions need such different versions, and led to quite extensive changes to the way units are stored, with new Monoid and Non-Monoid STS definitions, which all dimensions are assigned one or other of, in place of the old PhysicalDimension. For more details on this change, see Section 4 of H.

8.2 Other Changes

It was also decided that, besides examples like Fahrenheit, where using an OMF is incorrect, all of the units where the OMF used is "architected"[1] should be replaced by elements of **Q**. Therefore, this was carried out for every OMF that was deemed to be architected.

8.2.1 Definitions Added

It was found that units_imperial1 was missing several units, such as inch, stone and gallon, which it was felt should be present. These were duly added. In addition, the support for U.S. units appeared to be limited, so several new units were added. Finally, a problem was noticed whereby the unit Watt was defined to have the dimension `power` in `dimensions1`, but this particular symbol did not exist, so it was added.

[1] This term, and other related terms, are explained in the paper in Appendix H.

8.2.2 Units Moved

As a result of the discussion in H, `litre_pre1964` was moved to a new units_metric_obsolete1 CD. Additionally, a `units_imperial_obselete1` CD was created, with the units rod, pole, perch, chain and league

8.2.3 Units Removed

There were several compound units in the CDs, and, as previously stated, these were removed. However, where such a compound unit has a specific name, such as litre being a variation of metre_cubed, and acre being a variation on yard_sqrd, these were left in, as otherwise the system would not be able to recognise them, and they also required an additional multiplier.

Chapter 9

Conclusions

9.1 Introduction

What began as an implementation-based project developed into an investigation to determine what can be achieved in the area of unit conversion using OpenMath, with less emphasis on our actual implementation. We learnt a lot about the difficulties of unit conversion, and encountered a variety of hurdles. Many of these we managed to overcome, but several remain. Some involve OpenMath itself, while others are specific to our system or are more general to unit conversion. Overall, the project showed that unit conversion using OpenMath is feasible, but further work is needed to develop OpenMath's support for units in order to improve the functionality. There are also various areas in the system developed that could be improved, both for efficiency reasons and to increase the number of conversions possible. Firstly, we will examine the conclusions for OpenMath.

9.2 OpenMath

9.2.1 Conclusions for OpenMath

The project showed that OpenMath can be used for unit conversion. However, to fulfil its potential, several new features need to be considered and implemented. The main need is for mechanisms to determine whether a unit can take prefixes and what measurement standard the unit is. The current system attempts to guess both of these—prefixability is very difficult to predict algorithmically, however, and measurement standard can only be determined by the file name. This, however leads to some unanswered questions—supposing a method is found to specify whether a unit is imperial, time, and so on, it is not clear what should happen when units are combined. Firstly, how is it determined what the standard is for a unit that is, for example, imperial divided by time? And the reverse question also applies—if the user requests a speed unit to be expressed in imperial, how are the units for this chosen? How building any of this into OpenMath could be achieved is currently unknown. A possible method would be to add a new type to OpenMath, purely for units, which would allow additional tags (in the XML format) for these fields. It is not possible to tell at this stage whether this proposal is a reasonable one. The most important point is that options are fully considered before any decision is made, especially if it involves changing the OpenMath standard. It is also necessary to be able to determine whether the unit is a current unit, or an obsolete unit.

Another area that needs work is that of contracted unit names of three kinds—pseudonyms, unit symbols and abbreviations. It is not clear whether unit symbols will ever be a part of OpenMath—they may belong outside of it. It should be possible to have a set of pseudonyms for a unit, including plurals and similar; however it is unclear to what extent this is an OpenMath problem and to what extent it should be solved in software. A possible part of this, for plurals, would be to store a field in the suggested unit type, which contains the name of the pluralised form of a unit. In some respects the feasibility of using OpenMath for unit conversion is limited by the lack of abbreviations and so on, because it

severely limits the user-friendliness of the system.

Finally, it has been observed that there are no displacement units or velocity units existing in OpenMath at present. There is currently no method to distinguish a displacement from a distance, and thus a velocity from a speed. When there are units for velocity as well, it is uncertain what will happen with acceleration, because acceleration is rate of change of velocity, but is commonly used for rate of change of speed as well. Should a new acceleration dimension be produced, or will the old one suffice? This is another issue that has not been decided as yet.

9.2.2 Changes to OpenMath CDs

It seems reasonable to remove the "us" part from the unit names in `units_us1`, because it is somewhat redundant. In addition, all the suggestions in Appendix H regarding CDs should be considered, where they have not already been implemented.

9.3 Conclusions for the Project

The system produced was found to be very effective at many kinds of conversion. However, although not intentional, the system is only usable as a single-user system, because, although it met its requirements by taking less than 5 seconds to perform any conversion, this is still quite a long time, and if several users try to run the converter at the same time, problems could ensue with the server potentially becoming overwhelmed rapidly.

Although the method for loading unit definitions in seems to be a good idea, because it ensures that all the units definitions are the latest available, it means that conversions happen very slowly. Even though it only takes a couple of seconds, this prevents the system from being able to handle multiple requests simultaneously. The web server spawns several copies of the program, but, possibly because they all access the same files, the system execution time is increased.

There are some aspects of the system which initially really prevented it from being a multi-user system. The main one was that, if a user uploads an invalid CD, they would be informed, but the file would remain there for 30 minutes, causing any conversions run in that time to fail. The system now detects the invalid CD errors, and deletes the offending file. Disabling the network connection also speeds up the conversion incredibly, which shows that the cause is not in the program itself but in the network speed. Of course, for a web-based system, the network connection is necessary, but this demonstrates the root of the problem.

At an early stage in the processing by the system, the user's input gets converted to lower case, for ease of comparisons. However, this potentially leads to problems with units such as the calorie, which, when capitalised to Calorie, in fact means the kilocalorie. Therefore, it is felt that this mechanism should be changed because it reduces the correctness of the

system.

9.4 Unit Expectation

As noted in appendices F and H, converting "50 miles to metric" comes out as ~ 0.08 megametre, which, from a user's point of view, is unexpected. This is because there is a concept of the "preferred unit". In some units, the "kilo-" prefix is preferred for that magnitude and above, such as for metre. It does not matter how many hundreds or thousands of metres a length is, it would still be expected to return a result in kilometres. However, this is not a universal rule, as with mass, "tonne" is preferred for "megagramme" and above. If this were to be implemented in a system, it will need several issues resolving to go forward. One is the main issue of having a method in OpenMath to determine which units can be prefixed, but a mechanism for choosing appropriate units from a preference list would be required. This would, we assume, have to be written by a human, rather than determined programmatically, because there is no fundamental logic to it. We have not found any literature regarding this issue, but this may be because we also did not find any converters with the feature to convert to a measurement standard, and therefore maybe it has not been considered.

9.5 Further Work and Possible Extensions

One of the main tasks that needs to be undertaken is resolving the known problems. The main problem here is that of finding a method that will allow conversion of both acre-foot using a system-generated definition, and still allowing other conversions to work. This seems to require an algorithm to determine whether units should be split or combined, and it is felt that it should be possible to create such an algorithm based on some features of the units involved. Also, there are problems involving conversions with some prefixed units, which will need resolving.

Once the algorithm has been perfected, the acre-foot definition in units_imperial1 will no longer be necessary.

The system has already garnered some interest from the OpenMath community, with several people seeming very interested in how the system is panning out, and one member enquiring about a particular practical use to which they would like to apply the system. This novel use is as follows. The correspondent is involved in writing a "reasoner" of metric units, which asks the user a question and then attempts to reason about why the user gave the answer they did. As part of this, for example, the reasoner might be expecting the answer "0.5 kilometre", but receive the answer "500 metres", and one of the applications of our system in this is that it could be asked for "0.5 kilometre in metre", and thus inform the reasoner that the two results are indeed the same. An even more interesting extension the correspondent enquired about, and we feel is entirely feasible, was also reasoning about

why the user provided an answer, but this time the answer was actually incorrect, rather than just in a different form. The example supplied was that of saying 1000 seconds in minutes is 10 minutes, and the reasoner would be required to work out that the user had incorrectly assumed that a minute was 100 seconds. To use our system in this, truly an application that no other web-based converter can be used for, due to its extensibility, the reasoner could attempt to guess what the user thought based on the difference between their answer and the correct one, and then it could provide our system with a definition of a "buggy_minute", equal to 100 seconds, attempt the conversion, and receive the result that demonstrates that the user did indeed make this mistake, or that they made a different one. This would, however, require some minor modifications to the system. It is unlikely that the reasoner program would use the web-based front-end as the interface to the converter, and therefore some other mechanism for uploading the CDs would be required, and a method for the reasoner (or any other such system) to run the back-end and retrieve the result would need to be established.

Another part of the system that needs developing is allowing combinations of units, such "1 foot 2 inches 3 mils" as the source unit of the conversion. It should not be too difficult to implement—prior to starting the requested conversion the combination units would presumably be converted into a single unit themselves, for example, in this example we might get 14003 mils. This is a different problem to that of compound units, which only have a single quantity.

9.6 Recommended Changes to the System

There are some changes that are relevant only to the system, and some that are relevant only to OpenMath. However, there are still more for which a decision in one (usually OpenMath) necessitates a decision in the other.

A problem, which is more related to OpenMath's implementation of units is the prefixing problem, which currently uses the IsSimple method in our OpenMathUnit class, but has several hard-coded cases to cover all possibilities.

It has been noted that both OpenMathSimpleType and OpenMathCompoundType are not really used, so could probably be removed.

As was noted, several aspects of the front-end did not pass the usability and accessibility requirements. These should be resolved.

9.6.1 Bug Fixes

There are several problems with the system which mean that it does not successfully convert every conceivable unit.

One is that we currently have no good answer for how to use an FMP which contains an interval, such as calendar_year's 365-366 days. This is explored in section 5.1 of H.

Something that needs considering in OpenMath, but which has quite wide-ranging consequences for our system is whether it is correct for a unit name to appear in two different CDs, both to represent the same unit, as with second, or a different unit. This affects how the units are stored and looked up.

9.6.2 Non-essential Improvements

A major issue with the system as it stands is that of its runtime. This might be classed as an essential improvement if the system were to be used in the real world, particularly for multiple users. Because it looks on the Internet every time the converter is run, even though it usually does not download any files, the time taken for this is highly dependent on the internet connection available. This means that the system should be modified to go back to the original design plan, which was to download the files periodically with one program, and then use these local file exclusively in the system itself. This would involve a scheduler looking on the OpenMath website periodically, and the frequency of these checks would need to be decided. It is unclear how often the OpenMath website may be updated with files, as during the course of the project, the unit files were not updated because they were waiting for all the changes as a result of the system being implemented before being uploaded. Once they are there, it might be reasonable to check the website as infrequently as once a week, for example, but this might not be often enough, and therefore there should probably be a control on the user interface to instruct the update program to run immediately. As we have seen, this would only take a matter of seconds. The current problem with the speed means that the system as it stands, although web-based, is too slow to handle multiple users simultaneously. As a very small-scale and unrealistic test of this, we opened 10 copies of the front-end, and ran 10 different conversions simultaneously. Even though these were run on the same system as the back-end, and therefore most of the network activity which would normally be required was eliminated, it took between 5 and 10 seconds for all the results to be returned, which is really too long to be usable.

If the system were to be used as a serious converter, it would need a significant redesign in some areas. As previously stated, if the user uploads a CD with a unit that shares the name of another, which unit then gets chosen is difficult to determine. This is exacerbated if multiple users are using the system, uploading CDs and attempting to interact with CDs that they are not even aware of. This leads to two considerations. One is that if a user uploads a file for temporary use, it should only be usable by them; if they want other users to have access to it, they can either provide it themselves for a small number of users, or submit it to the OpenMath community.

There are a whole range of minor improvements, which were not deemed important enough to fix in the time available. One problem we attempted several times to solve was trying to validate the CDs against the CD schema. The problem here seemed to be caused by a problem with the schema, but one or two members of the OpenMath community who were consulted about this did not know what the cause was, and the problem was relegated to the list of further work, because it did not seem sufficiently important to justify spending

a lot of time on it. Another possible set of improvements comes by way of the fact that several of the variables stored in OpenMath can only have a specific set of values, such as status in the CD itself, and the Role in each OMS definition. Our system stores these as strings, but because the set of values is restricted, it would be reasonable for them to be stored as enumerations. This was not changed in the system because neither variable was used by the system, just stored.

An additional speed increase could be achieved during the ConvertToMetric method. As the units are stored in a HashTable, which is not in a logical order, this means that all the prefixed versions have to be attempted before determining which is best. However, it would be more efficient to iterate through the prefixes in order of size and stop when the one with the smallest absolute log has been found.

It would be desirable to have a better mechanism for the user for combining units. The system currently requires the user to spell out laboriously the desired unit, for example "metres per second sqrd". This is because the only changes made by the front-end are to insert _ as required. It could be possible for the front-end of the system to recognise operators, and simple calculations, for example "(2+3) miles/hour$\hat{2}$" and either pass this to the back-end or pass an equivalent request, having simplified it. Certainly, operators should be allowed instead of "per" and "sqrd".

At some stage, the log operator should be implemented, to allow ratio conversions to work. This would not be difficult, just mapping the symbol to a C# function. This has not happened so far because no units with logs have been encountered in the system.

Another area of inefficiency is the code to replace OpenMathSymbol objects with OpenMathOperator objects as appropriate. It came out of an early development that replaced all symbols with units or operators, and therefore it is not very efficient for its current use. It could be rewritten so that every symbol that is read is replaced by an OpenMathOperator as necessary during the loading stage.

It is felt it should be possible for the user to choose how non-metric standard conversions are output—rather than a long string of potentially many units, they should be offered the choice of getting an output comparable to that of the ConvertToMetric method.

9.7 Remaining Questions

Besides those already mentioned, there are a few further questions. One is whether the system should use the CDs in the "contrib" section on the OpenMath website. These are not official, and therefore so far have not been used.

As stated, at present, units with two FMPs cannot be supported fully. The question remains how to improve on this. A possibility is to use OpenMath's "kind" attribute to state, for example whether an FMP is definitional or not. Then the system could choose the most appropriate. This does not help the case where all the FMPs are needed and cannot be chosen between.

9.8 Final Thoughts

It is felt that the system, and other areas of the project, have significantly advanced the knowledge of using OpenMath for unit conversion, and the system itself was for the most part a success. It is recognised that this report is not the final word, so suggestions have been offered to direct and inspire future work in the area.

Bibliography

Belgian Ministry of Economic Affairs (1970), *Tableau Fixant Les Unites De Mesures Legales Et Leurs Multiples Et Sous-Multiples Annexe A L'arrete Royal Du 14 Septembre 1970 Portant Mise En Vigueur Partielle De La Loi Du 16 Juin 1970 Sur Les Unites; Etalons Et Instruments De Mesure Et Fixant Les Unites De Mesure Legales Et Les Etalons Et Les Mesures Necessaires A La Reproduction De Ces Unites.*
URL: *http://mineco.fgov.be/organization_market/metrology/showole_FR.asp?cParam=1150*

Bell College (2006), 'Software development 2: Graph data structures', `http://hamilton.bell.ac.uk/swdev2/notes/notes_18.pdf`.

Bourret, R. (2007), 'XML data binding resources', `http://www.rpbourret.com/xml/XMLDataBinding.htm`.

Bureau International des Poids et Mesures (1901), 'Resolution 1 of the 3rd CGPM', `http://www.bipm.org/en/CGPM/db/3/1/`.

Bureau International des Poids et Mesures (1964), 'Resolution 6 of the 12th CGPM', `http://www.bipm.org/en/CGPM/db/12/6/`.

Bureau International des Poids et Mesures (1979), 'Resolution 6 of the 16th CGPM', `http://www.bipm.org/en/CGPM/db/16/6/`.

Buswell, S., Caprotti, O., Carlisle, D. P., Dewar, M. C., Gaëtano, M. & Kohlhase, M. (2004), *OpenMath Standard 2.0.*
URL: *http://www.openmath.org/standard/om20-2004-06-30/omstd20.pdf*

Buswell, S., Davenport, J., Carlisle, D. & Dewar, M. (2004), *OpenMath - Guidelines for Tool Developers.*
URL: *http://www.openmath.org/projects/thematic/tools-3.pdf*

Chan, T. M. (2006), 'All-pairs shortest paths for unweighted undirected graphs in o(mn) time', *Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm* pp. 514–523.

Chan, T. M. (2007), 'More algorithms for all-pairs shortest paths in weighted graphs', *Proceedings of the thirty-ninth annual ACM symposium on Theory of computing STOC '07* pp. 590–598.

Davenport, J. H. (2000*a*), 'A Small OpenMath Type System', *ACM SIGSAM Bulletin* **34**(2), 16–21.

Davenport, J. H. (2000*b*), 'On Writing OpenMath Content Dictionaries', *ACM SIGSAM Bulletin* **34**(2), 12–15.

Davenport, J. H. & Naylor, W. (2003), *Units and Dimensions in OpenMath*.

de Halleux, J. (2007), *QuickGraph*.
URL: *http://www.codeplex.com/quickgraph*

Dijkstra, E. W. (1959), 'A note on two problems in connexion with graphs', *Numerische Mathematik* **1**(1), 269–271.

Johnson, D. B. (1977), 'Efficient algorithms for shortest paths in sparse networks', *Journal of the Association of Computing Machinery* **23**(5), 1–13.

Lange, C. (2008), *SWiM: A Semantic Wiki for Mathematical Knowledge Management*.
URL: *http://swim.kwarc.info/*

Mars Climate Orbiter Mishap Investigation Board (1999), *Mars Climate Orbiter: Phase I Report*.
URL: *ftp://ftp.hq.nasa.gov/pub/pao/reports/1999/MCO_report.pdf*

Microsoft Corporation (2008*a*), *double (C# Reference)*.
URL: *http://msdn2.microsoft.com/en-us/library/678hzkk9.aspx*

Microsoft Corporation (2008*b*), *long (C# Reference)*.
URL: *http://msdn2.microsoft.com/en-us/library/ctetwysk.aspx*

Nelson, W. H. (1997), 'The Gimli Glider', *Soaring Magazine* .
URL: *http://www.wadenelson.com/gimli.html*

Oxford English Dictionary (2007*a*), 'Entry for cubit', `http://dictionary.oed.com.ezp1.bath.ac.uk/cgi/entry/50055298?queryword=cubit`.

Oxford English Dictionary (2007*b*), 'Entry for unit', `http://dictionary.oed.com.ezp1.bath.ac.uk/cgi/entry/50267791?queryword=unit`.

Seidel, R. (1992), 'On the all-pairs-shortest-path problem', *Proceedings of the twenty-fourth annual ACM symposium on Theory of computing STOC '92* pp. 745–749.

Skiena, S. (1997), 'Data structures for graphs', `http://cs.sunysb.edu/~algorith/lectures-good/node14.html`.

Wagner, R. A. (1976), 'A shortest path algorithm for edge-sparse graphs', *Journal of the Association of Computing Machinery* **23**(1), 50–57.

Williams, M. (2003), 'The 156-tonne Gimli Glider', *Flight Safety Australia* .
URL: *http://www.casa.gov.au/fsa/2003/jul/22-27.pdf*

World Wide Web Consortium (1999), 'Web content accessibility guidelines 1.0', `http://www.w3.org/TR/WCAG10/`.

World Wide Web Consortium (2000*a*), 'CSS techniques for web content accessibility guidelines 1.0', `http://www.w3.org/TR/WCAG10-CSS-TECHS/`.

World Wide Web Consortium (2000*b*), 'Techniques for web content accessibility guidelines 1.0', `http://www.w3.org/TR/WAI-WEBCONTENT-TECHS`.

Zwick, U. (1998), 'All pairs shortest paths in weighted directed graphsexact and almost exact algorithms', *Proceedings of the 39th Annual IEEE Symposium on Foundations of Computer Science* pp. 310–319.

Zwick, U. (2002), 'All pairs shortest paths using bridging sets and rectangular matrix multiplication', *Journal of the ACM* **49**(3), 289–317.

Appendices

Appendix A

System Designs: Back-end

A.1 Original Design

This section details the original design for the system.

A.1.1 Class Functionality

OpenMath

This will be the base class for the system, containing any functionality common to all the OpenMath classes.

OpenMathCD

This will store all the data relating to a particular Content Dictionary, such as its name and version, and of course the definitions stored within it.

OpenMathSimpleType

This will be a "sub-base" class for the simple OpenMath classes, such as OpenMathInteger and OpenMathFloat.

OpenMathVariable

This stores the name of an OMV, which for the purposes of this system will probably only be used in the dimensions STS.

OpenMathSymbol

An OMS has a symbol name and an associated CD, both of which this class will need to store.

OpenMathOperator

The proposed system could load the definition of all the operators, but this would overly complicate a system which, really, only needs to use eq in relation1, and times, divide, plus, minus and power from arith1 (it has been decided in Davenport & Naylor (2003) that these are the sensible choices for units). Therefore, we have decided to instead recognise this subset of operators, and replace the symbol with an OpenMathOperator for that object, so that in the algebra parsing parts of the design, the system can easily tell which operator to use, and act accordingly.

OpenMathInteger

This will store the value of the integer. As OpenMath integers are infinite precision, and programming language ones are not, it will use the highest precision integer type available in C#—`long`, which has a range of -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807, and is the largest signed integer type available (Microsoft Corporation 2008*b*).

OpenMathFloat

This will store the value of the floating point number. As OpenMath floats are infinite precision, and programming ones are not, it will use the highest precision floating point type available in C#—`double`, which has an approximate range of $\pm 5.0 \times 10^{-324}$ to $\pm 1.7 \times 10^{308}$, and a precision of 15–16 digits (Microsoft Corporation 2008*a*).

OpenMathCompoundType

This will be a "sub-base" class for the OpenMath classes which can contain others, such as OpenMathApplication and OpenMathDefinition.

OpenMathDefinition

This is a superclass for OpenMathDimension and OpenMathUnit, but can also be instantiated to store a prefix, for example.

OpenMathUnit

This is a subclass of OpenMathDefinition for storing unit data.

OpenMathDimension

This is a subclass of OpenMathDefinition for storing dimension data.

OpenMathSTS

This is an OpenMathCompoundType which stores STS information for any other class that require it

OpenMathApplication

This is a class which will store the content of an OMA, and provide functionality to access/modify that content.

Graph

This class will contain a list of OpenMathUnits and a list of Edges between those units.

Edge

This class will store the two OpenMathUnits it connects, and the relevant FMP to move between the two.

A.1.2 Class Hierarchy

Based on this description, the class hierarchy shown in figure A.1 on page 95 was devised.

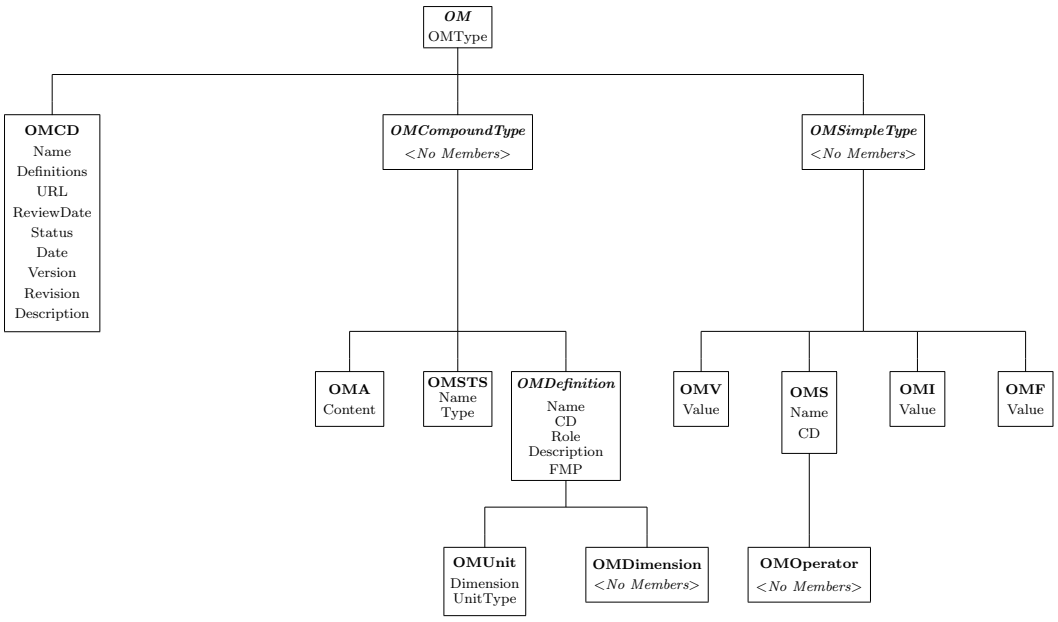


Figure A.1: The original Class Hierarchy for the system's data storage classes.
Key: **Class Name**, ***Abstract Class Name***, Members, <*No Members*> denotes no members.

A.2 Intermediate Design

After this, it was realised that there was no need for the OMSTS class, as it was just connecting a single OpenMathDimension to an OpenMathUnit, and the effect could be achieved more simply with just an extra field in the OpenMathUnit class. Therefore, the class was removed. The class hierarchy of this design is show in figure A.2 on page 97. There were no other changes to this intermediate design.

After this, implementation was started, although changes to the design were still permitted. This turned out to be fortunate, as during the implementation it was realised that a few simple changes would result in a system that was significantly easier to implement, as will be discussed in the following section.

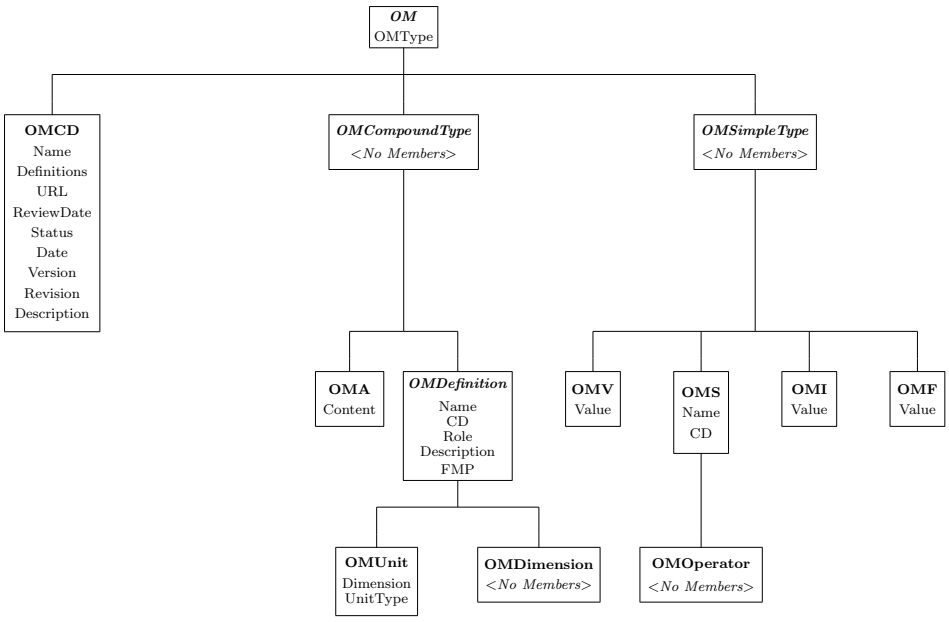


Figure A.2: An old Class Hierarchy for the system's data storage classes.
Key: **Class Name**, ***Abstract Class Name***, Members, <*No Members*> denotes no members.

A.3 Final Design

The final change to the hierarchy was to add an OpenMathNumericType class, as a superclass to OpenMathInteger and OpenMathFloat and a subclass to OpenMathSimpleType, for convenience of implementing the various operators—without the new class, every operator required 4 comparison cases for the combinations of OMI and OMF, and another set of cases for if one operand was an OMA. With the new class, this was reduced to one case covering the four combinations, and two for the OMA case, as it meant that it was only necessary to check both operands were this OpenMathNumericType class, rather than if each operand was an OpenMathInteger or an OpenMathFloat. This new class contained a floating point variable of type `double`, which could serve the purpose of storing either the `double` for an OpenMathFloat, or the `int` for an OpenMathInteger. It unfortunately meant that the system would be doing floating point arithmetic more often, which is inherently less accurate for very large numbers and recurring decimals, but it was felt that that the loss of accuracy was worth the simplification it granted, as it meant that the system was likely to have fewer bugs. In this section, we will give much greater detail regarding the proposed classes, as this is the final design.

This hierarchy only details the classes specifically related to OpenMath objects. In this final design, a further class was also added to the "additional classes", which previously consisted of Graph and Edge: the Conversion class.

A.3.1 Data Structures

All of the data will be stored in several classes. This section will describe the data storage classes used for all of the unit data.

For each class, a summary of the reason for its existence will be followed by a table specifying any data stored within that class, and this will be followed by a description of any methods in the class.

Each instantiable (that is, non-abstract) class in this hierarchy will override the default equality operators, so that if two such objects contain the same information, they are considered to be equal. They will also contain a method, Get(), to return an identical copy of themselves. If a class contains any additional methods, these will be explained in this section.

OpenMath

This will be the (abstract) base class for the system, containing any functionality common to all the OpenMath classes. It also contains a variable which specifies the type of OpenMath object, for ease of identifying. This field is not strictly necessary, because it is possible to determine the class type by other means; however, in some cases it is easier to use this method rather than another.

OpenMath	
Datum	**Type**
Type	Enum: {OMA, OMI, OMS, OMV, OMF, Dimension, Definition, Unit, Operator}

This class contains no additional functionality.

OpenMathCD

This will store all the data relating to a particular Content Dictionary, such as its name and version, and of course the definitions stored within it.

OpenMathCD:OpenMath	
Datum	**Type**
Name	String
Date	DateTime
ReviewDate	DateTime
Definitions	OpenMathDefinitions Array
Description	String
Version	Integer
Revision	Integer
Status	String
URL	String

This class contains the following method:

- A Contains method, which determines whether the specified OpenMathSymbol is contained in this CD.

OpenMathCompoundType

This will be an abstract "sub-base" type for the OpenMath types which can contain others, such as OpenMathApplication and OpenMathDefinition.

OpenMathCompoundType:OpenMath	
Datum	**Type**
<No Members>	

This class contains no additional functionality.

OpenMathApplication

This is a class which will store the content of an OMA, and provide functionality to access/modify that content, for example to simplify fractions containing only numbers to an OpenMathNumericType.

OpenMathApplication:OpenMathCompoundType	
Datum	**Type**
Content	OpenMath Array

One of the more complex classes, the OpenMathApplication class contains the following methods

- An Add method, to add an item to the OMA.
- A Simplify method, which reduces the OMA to its simplest form
- An Unwrap method, which, when applied to an OMA with a single OMS, unravels the rest of the OMA to obtain a numeric value
- A ReverseApplyTo method, which reverses the effect of an OMA and applies it to a particular value
- A FindUnits method, to find all the units within the OMA
- Several Replace methods, which replace all occurrences of the first parameter with the second parameter
  - `OpenMathUnit` to `double`, which creates an OpenMathFloat to store the `double` value
  - `OpenMathSymbol` to `OpenMath`
  - `OpenMathVariable` to `OpenMathSymbol`
- A GetAll method, which returns all of the objects of the specified type in the OMA
- A ReplaceFirst method, which replaces the first occurrence of a particular symbol with the given OpenMath object

OpenMathDefinition

This is a superclass for OpenMathDimension and OpenMathUnit, but can also be instantiated itself. It contains all the data that can be obtained reading a particular definition in a CD.

OpenMathDefinition:OpenMathCompoundType	
Datum	**Type**
Name	String
CD	String
Role	String
Description	String
FMP	OpenMathApplication

This class contains no additional functionality.

OpenMathUnit

This is a subclass of OpenMathDefinition for storing unit data. It contains additional flags and values that only units can have. The "None" value for unit type is available for units where the type could not be determined—at present there is no guaranteed method for determining this—but also means that more generic methods can be written, that will process units as well as other kinds of definition.

OpenMathUnit:OpenMathDefinition	
Datum	**Type**
Dimension	OpenMathDimension
Type	Enum: {Metric, Imperial, Time, US, None}
Prefixable	Boolean
Abbreviations	String Array

This class contains several methods:

- An AllAbbreviations method, which returns a list of all the abbreviations
- A ConvertSymbolsToUnits method, which converts an array of OpenMathSymbol to an array of the corresponding OpenMathUnits
- A GetConstituentUnits method to return an array of the first level of units that make up this unit (i.e. non-recursive)
- An IsSimple method, which returns whether the unit is one of a basic type or not

OpenMathDimension

This is a subclass of OpenMathDefinition for storing dimension data. Although it contains no values, it is useful to be able to tell it apart from its superclass.

OpenMathDimension:OpenMathDefinition	
Datum	**Type**
<No Members>	

This class contains no additional functionality.

OpenMathSimpleType

This will be an abstract "sub-base" type for the simple OpenMath types such as OpenMathVariable.

OpenMathSimpleType:OpenMath	
Datum	**Type**
<No Members>	

This class contains no additional functionality

OpenMathVariable

This stores the name of an OMV, which for the purposes of this system will only be used within the dimensions STS.

OpenMathVariable:OpenMathSimpleType	
Datum	**Type**
Name	String

This class contains no additional functionality.

OpenMathSymbol

An OMS is uniquely identified by a symbol name and an associated CD name, both of which this class will store.

OpenMathSymbol:OpenMathSimpleType	
Datum	**Type**
Name	String
CD	String

This class contains no additional functionality.

OpenMathOperator

This is not really an OpenMath type; however, the designed implementation of the system is easier if the operators are a subclass of OpenMathSymbol. Due to the reasons put forward in section 4.4.2, we have decided to recognise this subset of operators, and replace the OMS with the corresponding OpenMathOperator for that object, so that when parsing the algebra, the system can easily tell which operator to use, and act accordingly.

OpenMathOperator:OpenMathSymbol	
Datum	**Type**
<No Members>	

The operator objects will be singletons stored as static fields in this class, one for each of the recognised operators—`eq` from `relation1`, and `times`, `divide`, `plus`, `minus` and `power` from `arith1`. This class will also define the functionality of each operator, and contains the following methods:

- A GetOperator method, which returns the singleton OpenMathOperator corresponding to the supplied name and CD, or null.
- An IsKnownOperator method, which returns true if the supplied name and CD are that of a known operator, false otherwise
- An Operate method which performs the operation specified in an OMA, and returns the result, making use of the following methods:
  - opDivide
  - opEquals
  - opMinus
  - opPlus
  - opPower
  - opTimes

OpenMathNumericType

This is an abstract superclass for OpenMathInteger and OpenMathFloat added so that most parts of the system only need know that one or other is present, not specifically which one.

OpenMathNumericType:OpenMathSimpleType	
Datum	**Type**
Value	Float

This class contains no additional functionality

OpenMathInteger

This will store the value of the integer. Originally, the intention was that, as OpenMath integers are infinite precision, and programming language ones are not, it would use the highest precision integer type available, which in the case of C# is a `long`. However, to make implementation easier, its value is now stored in the `double` of OpenMathNumericType.

OpenMathInteger:OpenMathNumericType	
Datum	**Type**
<No Members>	

This class contains no additional functionality.

OpenMathFloat

This will store the value of the floating point number. Originally, the intention was that, as OpenMath floats are infinite precision, and programming language ones are not, it would use the highest precision floating point type available, which in the case of C# is a `double`. While this effectively still happens, the variable for storing the value is now located in OpenMathNumericType.

OpenMathFloat:OpenMathNumericType	
Datum	**Type**
<No Members>	

This class contains no additional functionality.

A.3.2 Class Hierarchy

Based on these descriptions, we have come up with the class hierarchy shown in Figure A.3 on page 104.

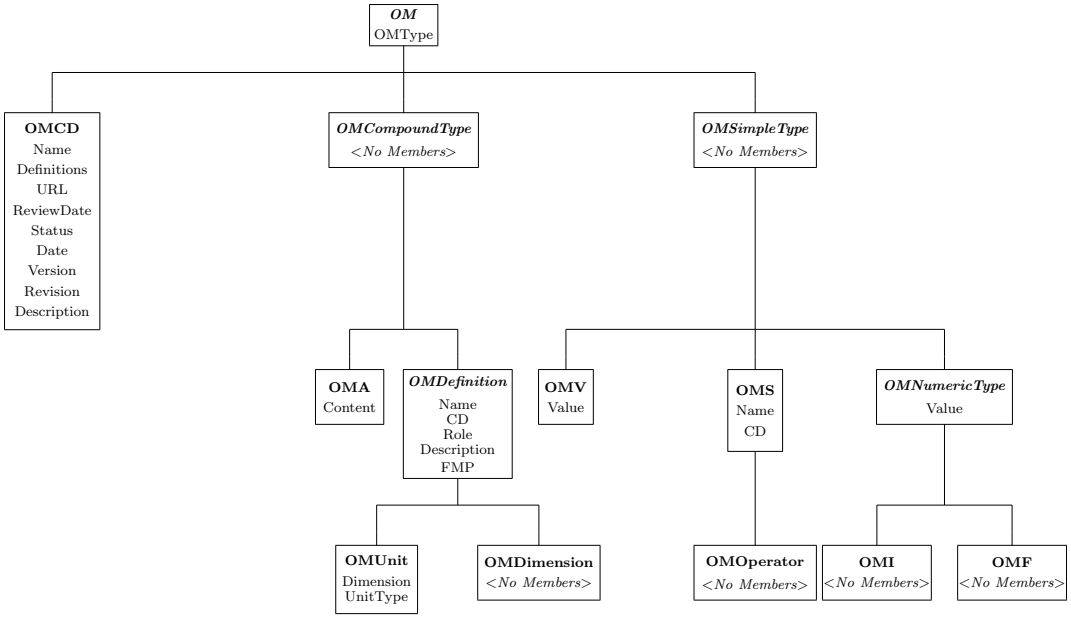


Figure A.3: The final Class Hierarchy for the system's data storage.
Key: **Class Name**, ***Abstract Class Name***, Members, <*No Members*> denotes no members.

A.3.3 Other Classes

Outside of this hierarchy, there are three further classes. Previously, there were only two, but a Conversion class has now been added, to logically group a lot of functionality regarding the conversion.

Graph

This class will contain a list of OpenMathUnits and a list of Edges between those units. A Graph object will be associated with each OpenMath dimension, and will only contain units that are of that dimension.

Graph	
Datum	**Type**
Dimension	OpenMathDimension
Edges	Edge Array
Units	OpenMathUnit Array

The Graph class contains functionality for finding a route between two of its units by way of the list of edges it has, as well as determining whether a unit is in this graph. For efficiency, units are stored internally in a HashTable, but the public interface offers an array, so that its contents cannot be modified.

Its list of methods is as follows:

- An Add method, for adding a new unit to the graph
- Three Contains methods, for determining whether a unit is in the graph, with the choice of OpenMathUnit,OpenMathSymbol or String representations of the unit
- An EdgesContaining method, which returns an array of edges which start or end with the specified unit
- A GenerateEdges method, which constructs Edge objects between OpenMathUnit objects where the definition of one refers to the other
- A GetIndex method which returns the index of the unit in the Units array of the graph, or -1 if not present
- A GetKeysArray method, which converts the HashTable unit keys to an array
- A GetShortestPath method, which attempts to find the shortest route[1] between the two specified units. It returns a list of edges representing the route, which is empty if the two units are the same, or null if a route could not be found between the two units.
- A GetSimpleUnitsOf method, which returns the simple units in the given unit type

[1] or "a reasonable short route"

- A GetUnitsCalled method, which returns a list of all the units with the given name—a graph can contain several units with the same name if they have come from different CDs
- A GetUnitsOf method, which returns an array of all the units in the graph that are of the specified unit type
- A RecursiveFinder method, which examines an FMP to find the units therein, to determine which unit should be the other end of an Edge

Edge

This class will store the two OpenMathUnit it connects, and the relevant FMP to move between the two. It is created by a Graph object, and is purely a storage class—it performs no processing at all.

Edge	
Datum	**Type**
Unit1	OpenMathUnit
Unit2	OpenMathUnit
Conversion	OpenMathApplication

This class does not contain any other functionality.

Conversion

This class covers all aspects of converting between two units of the same dimension. It determines how to convert between the two units, and stores the input value and resulting output value.

Conversion	
Datum	**Type**
Source	OpenMathUnit
Destination	OpenMathUnit
Input	OpenMathFloat
Output	OpenMathFloat

The Conversion class has effectively two modes, but it was felt best to store them in the same class as when created it is not known which mode to use[2]. The mechanisms for converting between units will be explained in the Detailed Design chapter in Section 5.3.4 on page 52.

The class has the following methods:

- A CreateConversion method, which attempts to make a conversion between the two units
- A Perform method, which performs any necessary conversions for a given input

[2]even with a subclass, one has to be instantiated, so this approach hides as much of the complexity away as possible

- A PerformConversion method, which takes an input value and uses the determined route to transform it, storing the result in a specified output variable
- A Replacer method, which takes an array of units and an index of which one to split further, returning a larger array with the specified unit split into its constituent parts, and the specified OpenMathApplication modified such that the original unit is replaced by its constituent parts in the same way the array has been

In addition to these three classes, there is also the main program class.

Program

This class contains many miscellaneous methods required to glue the whole system together.

Program	
Datum	**Type**
KnownCDs	OpenMathCD Array
OMType	Enum: (NoError, UnknownUnit, InvalidCommandLine, GraphsNotLoadable, UnitsDifferentDimensions, ConversionNotFound, ConversionFailed, UnitTypeNotFound, UnknownDimension, DuplicateUnit, DuplicateCd, XMLError)

The methods in this class are as follows:

- A Main method, the entry point to the system
- A ConvertToMetric method, which returns the best metric unit for the result, including prefix as appropriate
- A ConvertTo method, which returns the best unit(s) in the specified standard for the result, including prefix as appropriate.
- A LookUpUnit method, which returns a unit given a unit name. If the unit is not found in any of the graphs in the HashTable, this method will attempt to find a compound unit, potentially adding a new dimension to the HashTable of graphs, as appropriate. If the unit could still not be determined, null is returned.
- A GetDerivedUnit method, which is used by the LookUpUnit method to generate an appropriate compound unit from the unit name, which returns the unit it has generated, or null if it has failed to determine the unit.
- An AddAllPrefixedVersionsIf method, which, given a unit, a graph and an array of prefix definitions, generates all the prefixed versions of that unit, and adds it to the corresponding graph, before regenerating the edges for that graph
- An AddPrefixedVersion method, which adds a particular prefixed version of a specified unit to a specified graph.
- A ParseCommandLine method, which splits the (valid) command line into source quantity, source unit, and destination unit, which it returns in an ArrayList, or runs the PrintCommandLineDetails method, and return null if the syntax was incorrect.

- A PrintCommandLineDetails method, which displays an error message explaining what was wrong with the user's input, and how it should have been formed.
- A ReadUnitsIn method, which takes a URL to parse to find unit and dimensions CDs, then returns a HashTable of graphs for each of the different dimensions found, each containing all the units found of that dimension.
- A ReadDefsFromCDs method, which reads a set of a file names which are of a particular type (dimension, or unit, for example), and if a unit, a particular unit standard (metric, imperial etc), and also finds, reads and stores the STS dimension information if specified to, and returns a list of OpenMathCD objects read, and the list of definitions found. When working with units, it also requires the definitions of the dimensions known, for use with the STS information.
- A WriteNamesToFile method, which reads all the units out of the specified HashTable, and writes them, as appropriate, to two different files, one called units_prefixable.txt, and one called units_not_prefixable.txt, so that the front-end can load these in as suggestions to the user. Which file to put the unit in in determined by the Prefixable member in the OpenMathUnit class. In addition, the prefixes are written to the file prefixes.txt
- A ReplaceSymbols method, which replaces the OMSes in all the definitions in all the CDs passed to it, if they represent a known operator, with the OpenMathOperator equivalent
- A GetShortName method, which parses a URL or file name given to it, and returns the local file name without an extension (such that if passed the location of an OCD file, the name of the CD is returned. This assumes the name of a CD corresponds to its file name.
- A GetFileName method, which, given a CD name (as returned by GetShortName), will give back the address of either a CD or STS, depending on a boolean parameter
- A RecursiveOperatorReplacer which takes an FMP and replaces any OpenMathSymbol objects which represent a known operator with the corresponding OpenMathOperator.
- A ConvertToAbsolutePath method, which takes a path found in the file referred to by the URL passed to ReadUnitsIn (which will be a relative path to an XHTML file), and converts it to an absolute path to the OCD file using the specified base URL.
- A FindFileList method, which, given a list of identifiers (such as "dimensions" and "units_metric"), will return a 2D array of all the files found that start with each identifier, in the same order.
- A GetLocalFilename, which returns a relative file name to a local file, which is the latest available version of the remote file passed to it, having either downloaded the latest version, or established the the local version is at least as recent.

- A GetLocalStreamReader, which returns a StreamReader to a local file (as in GetLocalFileName), given a remote file name.
- A MakeGraphsFromUnits method, which generates a new graph for each new unit dimension found in a 2D array of all the units (each subscript of the first array dimension is a different unit type), and populates each graph with the relevant units
- A ReadSTS method, which reads the STS File for a given CD, and adds the corresponding dimension information from a given array of dimensions, to the given units
- An InitialiseXMLReader method, which takes a file name to an XML file (e.g. a CD) and returns an XMLReader with various properties set including a file-type-dependent schema.
- A ReadCD method, which takes the location of a CD of a specified type, (and unit type), and returns the definitions/dimensions/units stored in that CD, as well as adding the OpenMathCD to a global list of known CDs.
- An IsKnownCD method, which determines whether the specified CD is known to the system
- Various methods for reading the following from an XML file:
  - Definition
  - Example
  - FMP
  - OMOBJ
  - OMA

A.3.4 Public Interface

The design of the program also includes the command line arguments: --source_quantity, --source_unit, and --destination_unit. It is envisaged that these are all that is required for the system. Unit names for compound units are presently only recognised if they are of the form `metre_sqrd`, `yard_cubed`, `acre_foot`, or `mile_per_hour`, that is, with the words separated by _.

In addition, the following error codes will be defined, and for each, any text that is written to the standard error stream is listed:

0 NoError

1 UnknownUnit—the first unknown unit name written to standard error stream

2 InvalidCommandLine—command line details, and correct specification, written to standard error stream

3 GraphsNotLoadable

4 UnitsDifferentDimensions

5 ConversionNotFound

6 ConversionFailed

7 UnitTypeNotFound

8 UnknownDimension—the dimension name is written to standard error stream

9 DuplicateUnit—the unit and CD names are written to standard error stream

10 DuplicateCd—CD name written to standard error stream

11 XMLError—The specific problem found is written to standard error stream

Appendix B

System Designs: Front-end

B.1 Introduction

The front-end of the system does not have so much functionality, and therefore has less to it than the back-end. Indeed, it has been designed as an entirely self-contained single page. However, because it is the sole method of interaction with the system for the user, it is vitally important that it is done correctly. During the course of performing the analysis of other unit converters (Section 2.4, starting from page 6), a lot of insight was gained into what worked, and what did not, and these aspects were used to influence the design presented here.

B.2 User Interface Design

B.2.1 Features

This design features a fairly plain web page with two text boxes for input; one for the source quantity and unit, and one for the destination unit. In addition, there will be a drop box for choosing the number of significant figures, with the default set to the maximum number of significant figures that the back-end can return.

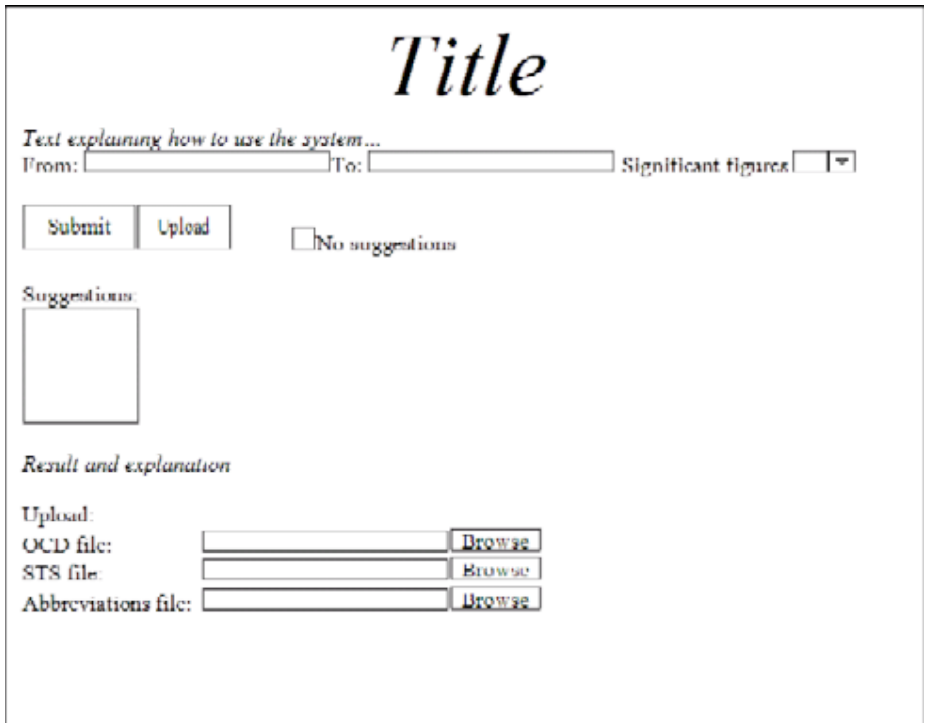
Then there will be two additional buttons, one which submits the form (also activated if <Enter> is pressed), and one for manually triggering the upload files mode—which is automatically initiated when a "unknown unit" error is returned from the back-end. This mode adds 3 file input boxes to the display, with labels for "OCD file", "STS file", and "(optional) Abbreviations file". When the user has selected these files and submitted the form, the files are uploaded. If the files were in response to the "unknown unit" error, additionally, the user's original request will be resubmitted. When in this mode, normal conversions can still occur, in case the user changes their conversion. Due to the requirements of usability, all the controls will have "label" tags, to allow screen-readers to identify them to visually-impaired users.

Below the conversion input, a list of suggestions will appear as the user types into one of the text boxes. These will be clickable, and clicking on one will result in the word the user is typing in the text box (which caused the suggestion) being replaced by the selected suggestion. It is also intended that the user can use the keyboard to access this list, with just the up and down cursor keys. Due to the dynamic nature of these, it may be that some users will not like this feature, particularly if they are using a screen-reader, so there will be an option to turn it off.

Below this, the output from the back-end will be displayed, with suitable accompanying text. This means that the user can update their original query while still seeing the result.

B.2.2 Layout of Design

Here is a mock-up of how the front-end is intended to appear. The "suggestions" section will only appear if there are suggestions to offer, and the Upload section will only appear in response to the "unknown unit" error code, or the user clicking the "Upload" button. As can be seen, the content of much of the text has been left to the implementer, as have more detailed design choices. It is open to the implementer whether the PHP populates a ready-made HTML template, or generates all the HTML dynamically, for example.



B.2.3 Functionality

Although the amount of functionality in the front-end will be kept to a minimum, there are a few functions that are necessary. One is the function for rounding to a certain number of significant figures. This will take the value and the number of significant figures to round it to, and return the value rounded to that number of significant figures.

Other functions will be needed to recognise suggestions and add the suggestion that has been clicked on to the correct box, replacing what the user was typing at the time.

Additionally, a function to ensure that the user's input is valid will be required, including that if they are trying to upload files that they have filled in the first two boxes (abbreviations file is optional), or none at all. This is so that the user does not have to wait until the

back-end has been called and an error has been returned to catch basic types of error, such as a missing destination unit. The front-end cannot stop the user entering a unit name that is not in the list of suggestions, because the user may be entering a compound unit, which would not show up, or additional files may have been provided to the back-end since the list was generated. For this reason, this design leaves all unit recognition to the back-end of the system.

B.3 Interface to Back-end

Although the front-end will not be processing a single input natural language field, it will still need to perform a bit of processing on the input. Firstly, it will have to separate out the quantity of the source unit, if present. This is fairly straightforward—the first "word" in the input, if numeric, is recognised to be the source quantity. It will then have to use the rest of the input from the source box as the source unit, and the entirety of the input from the destination box to mean the destination unit, having replaced any spaces in either with underscores. It will then prepend the flags to these values (see section A.3.4 on page 109), and call the back-end program with these options. Once it has received any output, it will read the error code returned, then, based on this, read the standard output stream or standard error stream as appropriate.

Appendix C

CDs

The CDs the have been changed as a result of the project appear in this Appendix

C.1 Modified CDs

C.1.1 File: dimensions1.ocd

```
<CD xmlns="http://www.openmath.org/OpenMathCD">

<CDComment>

    This document is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

    The copyright holder grants you permission to redistribute this
    document freely as a verbatim copy. Furthermore, the copyright
    holder permits you to develop any derived work from this document
    provided that the following conditions are met.
      a) The derived work acknowledges the fact that it is derived from
         this document, and maintains a prominent reference in the
         work to the original source.
      b) The fact that the derived work is not the original OpenMath
         document is stated prominently in the derived work.  Moreover if
         both this document and the derived work are Content Dictionaries
         then the derived work must include a different CDName element,
         chosen so that it cannot be confused with any works adopted by
         the OpenMath Society.  In particular, if there is a Content
         Dictionary Group whose name is, for example, 'math' containing
         Content Dictionaries named 'math1', 'math2' etc., then you should
         not name a derived Content Dictionary 'mathN' where N is an integer.
         However you are free to name it 'private_mathN' or some such.  This
         is because the names 'mathN' may be used by the OpenMath Society
         for future extensions.
      c) The derived work is distributed under terms that allow the
         compilation of derived works, but keep paragraphs a) and b)
         intact.  The simplest way to do this is to distribute the derived
         work under the OpenMath license, but this is not a requirement.
    If you have questions about this license please contact the OpenMath
    society at http://www.openmath.org.
</CDComment>

<CDName> dimensions1 </CDName>
<CDBase>http://www.openmath.org/cd</CDBase>
<CDURL> http://www.openmath.org/cd/dimensions1.ocd </CDURL>
<CDReviewDate>2008-03-31</CDReviewDate>
<CDStatus> experimental </CDStatus>
<CDDate>2004-03-30</CDDate>
<CDVersion>5</CDVersion>
<CDRevision>0</CDRevision>

<Description>
This CD defines symbols which represent basic physical dimensions.
</Description>

<CDDefinition>
<Name> length </Name>
<Role>constant</Role>
<Description>
This symbol represents the length physical dimension.
</Description>

</CDDefinition>

<CDDefinition>
<Name> area </Name>
<Role>constant</Role>
<Description>
This symbol represents the area physical dimension.
```

```
</Description>

<CMP> area = length*length </CMP>

<FMP><OMOBJ xmlns="http://www.openmath.org/OpenMath"
    version="2.0" cdbase="http://www.openmath.org/cd">
  <OMA>
    <OMS cd="relation1" name="eq"/>
    <OMS cd="dimensions1" name="area"/>
    <OMA>
      <OMS cd="arith1" name="times"/>
      <OMS cd="dimensions1" name="length"/>
      <OMS cd="dimensions1" name="length"/>
    </OMA>
  </OMA>
</OMOBJ></FMP>
</CDDefinition>

<CDDefinition>
<Name> volume </Name>
<Role>constant</Role>
<Description>
This symbol represents the volume physical dimension.
</Description>

<CMP> volume = length*length*length </CMP>

<FMP><OMOBJ xmlns="http://www.openmath.org/OpenMath"
    version="2.0" cdbase="http://www.openmath.org/cd">
  <OMA>
    <OMS cd="relation1" name="eq"/>
    <OMS cd="dimensions1" name="volume"/>
    <OMA>
      <OMS cd="arith1" name="times"/>
      <OMS cd="dimensions1" name="length"/>
      <OMS cd="dimensions1" name="length"/>
      <OMS cd="dimensions1" name="length"/>
    </OMA>
  </OMA>
</OMOBJ></FMP>
</CDDefinition>

<CDDefinition>
<Name> speed </Name>
<Role>constant</Role>
  <Description>
This symbol represents the speed physical dimension. It
    is the size of the
derivative of distance with respect to time.
</Description>

<CMP>speed = length/time</CMP>

<FMP><OMOBJ xmlns="http://www.openmath.org/OpenMath"
    version="2.0" cdbase="http://www.openmath.org/cd">
  <OMA>
    <OMS cd="relation1" name="eq"/>
    <OMS cd="dimensions1" name="speed"/>
    <OMA>
      <OMS cd="arith1" name="divide"/>
      <OMS cd="dimensions1" name="length"/>
      <OMS cd="dimensions1" name="time"/>
    </OMA>
  </OMA>
</OMOBJ></FMP>
</CDDefinition>

<CDDefinition>
  <Name> displacement </Name>
<Role>constant</Role>
  <Description>
This symbol represents the spatial difference between
    two points.
The direction of the displacement is taken into account
    as well as the
distance between the points.
  </Description>
</CDDefinition>

<CDDefinition>
<Name> velocity </Name>
<Role>constant</Role>
<Description>
This symbol represents the velocity physical dimension.
    It is the
derivative of distance with respect to time.
</Description>

<CMP> velocity = displacement/time</CMP>
```

```
<FMP>
    <OMOBJ xmlns="http://www.openmath.org/OpenMath"
        version="2.0"
        cdbase="http://www.openmath.org/cd">
      <OMA>
        <OMS cd="relation1" name="eq"/>
        <OMS cd="dimensions1" name="velocity"/>
        <OMA>
          <OMS cd="arith1" name="divide"/>
          <OMS cd="dimensions1" name="displacement"/>
          <OMS cd="dimensions1" name="time"/>
        </OMA>
      </OMA>
    </OMOBJ>
</FMP>

</CDDefinition>

<CDDefinition>
<Name> acceleration </Name>
<Role>constant</Role>
<Description>
This symbol represents the acceleration physical
    dimension. It is the
second derivative of distance with respect to time.
</Description>

<CMP>acceleration = displacement/(time^2)</CMP>

<FMP><OMOBJ xmlns="http://www.openmath.org/OpenMath"
    version="2.0" cdbase="http://www.openmath.org/cd">
  <OMA>
    <OMS cd="relation1" name="eq"/>
    <OMS cd="dimensions1" name="acceleration"/>
    <OMA>
      <OMS cd="arith1" name="divide"/>
      <OMS cd="dimensions1" name="displacement"/>
      <OMA>
        <OMS cd="arith1" name="power"/>
        <OMS cd="dimensions1" name="time"/>
        <OMI> 2 </OMI>
      </OMA>
    </OMA>
  </OMA>
</OMOBJ></FMP>

</CDDefinition>

<CDDefinition>
<Name> time </Name>
<Role>constant</Role>
<Description>
This symbol represents the time physical dimension.
</Description>
</CDDefinition>

<CDDefinition>
<Name> mass </Name>
<Role>constant</Role>
<Description>
This symbol represents the mass physical dimension.
</Description>
</CDDefinition>

<CDDefinition>
<Name> force </Name>
<Role>constant</Role>
<Description>
This symbol represents the force physical dimension.
</Description>

<CMP> force = mass*length/time^2 </CMP>

<FMP><OMOBJ xmlns="http://www.openmath.org/OpenMath"
    version="2.0" cdbase="http://www.openmath.org/cd">
  <OMA>
    <OMS cd="relation1" name="eq"/>
    <OMS cd="dimensions1" name="force"/>
    <OMA>
      <OMS cd="arith1" name="times"/>
      <OMS cd="dimensions1" name="mass"/>
      <OMA>
        <OMS cd="arith1" name="divide"/>
        <OMS cd="dimensions1" name="length"/>
        <OMA>
          <OMS cd="arith1" name="power"/>
          <OMS cd="dimensions1" name="time"/>
          <OMI> 2 </OMI>
        </OMA>
      </OMA>
```

```
    </OMA>
  </OMA>
</OMOBJ></FMP>

</CDDefinition>

<CDDefinition>
<Name> temperature </Name>
<Role>constant</Role>
<Description>
This symbol represents the "absolute" temperature
    physical dimension.
</Description>
</CDDefinition>

<CDDefinition>
<Name> relativeTemperature </Name>
<Role>constant</Role>
<Description>
This symbol represents the relative temperature physical
    dimension.
</Description>

</CDDefinition>

<CDDefinition>
<Name> pressure </Name>
<Role>constant</Role>
<Description>
This symbol represents the pressure physical dimension.
</Description>

<CMP> pressure = force/area </CMP>

<FMP><OMOBJ xmlns="http://www.openmath.org/OpenMath"
    version="2.0" cdbase="http://www.openmath.org/cd">
  <OMA>
    <OMS cd="relation1" name="eq"/>
    <OMS cd="dimensions1" name="pressure"/>
    <OMA>
      <OMS cd="arith1" name="divide"/>
      <OMS cd="dimensions1" name="force"/>
      <OMS cd="dimensions1" name="area"/>
    </OMA>
  </OMA>
</OMOBJ></FMP>

</CDDefinition>

<CDDefinition>
<Name> charge </Name>
<Role>constant</Role>
<Description>
This symbol represents the charge physical dimension.
</Description>

<CMP> charge = current/voltage </CMP>

<FMP><OMOBJ xmlns="http://www.openmath.org/OpenMath"
    version="2.0" cdbase="http://www.openmath.org/cd">
  <OMA>
    <OMS cd="relation1" name="eq"/>
    <OMS cd="dimensions1" name="charge"/>
    <OMA>
      <OMS cd="arith1" name="divide"/>
      <OMS cd="dimensions1" name="current"/>
      <OMS cd="dimensions1" name="voltage"/>
    </OMA>
  </OMA>
</OMOBJ></FMP>

</CDDefinition>

<CDDefinition>
<Name> current </Name>
<Role>constant</Role>
<Description>
This symbol represents the current physical dimension.
</Description>

<CMP> current = voltage*charge </CMP>

<FMP><OMOBJ xmlns="http://www.openmath.org/OpenMath"
    version="2.0" cdbase="http://www.openmath.org/cd">
  <OMA>
    <OMS cd="relation1" name="eq"/>
    <OMS cd="dimensions1" name="current"/>
    <OMA>
      <OMS cd="arith1" name="times"/>
      <OMS cd="dimensions1" name="voltage"/>
```

```
        <OMS cd="dimensions1" name="charge"/>
      </OMA>
    </OMA>
</OMOBJ></FMP>

</CDDefinition>

<CDDefinition>
<Name> voltage </Name>
<Role>constant</Role>
<Description>
This symbol represents the voltage physical dimension.
</Description>

<CMP> voltage = current/charge </CMP>

<FMP><OMOBJ xmlns="http://www.openmath.org/OpenMath"
    version="2.0" cdbase="http://www.openmath.org/cd">
  <OMA>
    <OMS cd="relation1" name="eq"/>
    <OMS cd="dimensions1" name="voltage"/>
    <OMA>
      <OMS cd="arith1" name="divide"/>
      <OMS cd="dimensions1" name="current"/>
      <OMS cd="dimensions1" name="charge"/>
    </OMA>
  </OMA>
</OMOBJ></FMP>

</CDDefinition>

<CDDefinition>
<Name> resistance </Name>
<Role>constant</Role>
<Description>
This symbol represents the resistance physical
    dimension, it is the
resistance that an electrical circuit has to flow of
    charge.
</Description>

<CMP> resistance = voltage/current </CMP>

<FMP><OMOBJ xmlns="http://www.openmath.org/OpenMath"
    version="2.0" cdbase="http://www.openmath.org/cd">
  <OMA>
    <OMS cd="relation1" name="eq"/>
    <OMS cd="dimensions1" name="resistance"/>
    <OMA>
      <OMS cd="arith1" name="divide"/>
      <OMS cd="dimensions1" name="voltage"/>
      <OMS cd="dimensions1" name="current"/>
    </OMA>
  </OMA>
</OMOBJ></FMP>

</CDDefinition>

<CDDefinition>
<Name> density </Name>
<Role>constant</Role>
<Description>
This symbol represents the density physical dimension,
    it is the mass
per unit volume.
</Description>

<CMP> density = mass/volume </CMP>

<FMP><OMOBJ xmlns="http://www.openmath.org/OpenMath"
    version="2.0" cdbase="http://www.openmath.org/cd">
  <OMA>
    <OMS cd="relation1" name="eq"/>
    <OMS cd="dimensions1" name="density"/>
    <OMA>
      <OMS cd="arith1" name="divide"/>
      <OMS cd="dimensions1" name="mass"/>
      <OMS cd="dimensions1" name="volume"/>
    </OMA>
  </OMA>
</OMOBJ></FMP>

</CDDefinition>

<CDDefinition>
<Name> energy </Name>
<Role>constant</Role>
<Description>
This symbol represents the energy physical dimension.
</Description>
```

```
<CMP> energy = mass*length^2/time^2 </CMP>

<FMP><OMOBJ xmlns="http://www.openmath.org/OpenMath"
    version="2.0" cdbase="http://www.openmath.org/cd">
  <OMA>
    <OMS cd="relation1" name="eq"/>
    <OMS cd="dimensions1" name="energy"/>
    <OMA>
      <OMS cd="arith1" name="divide"/>
      <OMA>
        <OMS cd="arith1" name="times"/>
        <OMS cd="dimensions1" name="mass"/>
        <OMA>
          <OMS cd="arith1" name="power"/>
          <OMS cd="dimensions1" name="length"/>
          <OMI> 2 </OMI>
        </OMA>
      </OMA>
      <OMA>
        <OMS cd="arith1" name="power"/>
        <OMS cd="dimensions1" name="time"/>
        <OMI> 2 </OMI>
      </OMA>
    </OMA>
  </OMA>
</OMOBJ></FMP>

</CDDefinition>

<CDDefinition>
<Name> concentration </Name>
<Role>constant</Role>
<Description>
This symbol represents the concentration physical
    dimension, it is the
amount of a substance in a volume.
</Description>

<CMP> concentration = mass/length^3 </CMP>

<FMP><OMOBJ xmlns="http://www.openmath.org/OpenMath"
    version="2.0" cdbase="http://www.openmath.org/cd">
  <OMA>
    <OMS cd="relation1" name="eq"/>
    <OMS cd="dimensions1" name="concentration"/>
    <OMA>
      <OMS cd="arith1" name="divide"/>
      <OMS cd="dimensions1" name="mass"/>
      <OMA>
        <OMS cd="arith1" name="power"/>
        <OMS cd="dimensions1" name="length"/>
        <OMI> 3 </OMI>
      </OMA>
    </OMA>
  </OMA>
</OMOBJ></FMP>

</CDDefinition>

<CDDefinition>
<Name> momentum </Name>
<Role>constant</Role>
<Description>
This symbol represents the momentum physical dimension,
    it is mass
times velocity.
</Description>

<CMP> momentum = mass*velocity </CMP>

<FMP><OMOBJ xmlns="http://www.openmath.org/OpenMath"
    version="2.0" cdbase="http://www.openmath.org/cd">
  <OMA>
    <OMS cd="relation1" name="eq"/>
    <OMS cd="dimensions1" name="momentum"/>
    <OMA>
      <OMS cd="arith1" name="times"/>
      <OMS cd="dimensions1" name="mass"/>
      <OMS cd="dimensions1" name="velocity"/>
    </OMA>
  </OMA>
</OMOBJ></FMP>

</CDDefinition>
<CDDefinition>
<Name> power </Name>
<Description>
This symbol represents the power physical dimension, it
    is energy
```

```
per time.
</Description>

<CMP> power = energy/time </CMP>

<FMP><OMOBJ xmlns="http://www.openmath.org/OpenMath">
  <OMA>
    <OMS cd="relation1" name="eq"/>
    <OMS cd="dimensions1" name="power"/>
    <OMA>
      <OMS cd="arith1" name="divide"/>
      <OMS cd="dimensions1" name="energy"/>
      <OMS cd="dimensions1" name="time"/>
    </OMA>
  </OMA>
</OMOBJ></FMP>

</CDDefinition>

</CD>
```

C.1.2 File: dimensions1.sts

```
<CDSignatures
    xmlns="http://www.openmath.org/OpenMathCDS"
    type="sts" cd="dimensions1">
<CDSStatus> official </CDSStatus>

<Signature name="length" >
<OMOBJ xmlns="http://www.openmath.org/OpenMath">
  <OMS name="MonoidDimension"/>
</OMOBJ>
</Signature>

<Signature name="area" >
<OMOBJ xmlns="http://www.openmath.org/OpenMath">
  <OMS name="MonoidDimension"/>
</OMOBJ>
</Signature>

<Signature name="volume" >
<OMOBJ xmlns="http://www.openmath.org/OpenMath">
  <OMS name="MonoidDimension"/>
</OMOBJ>
</Signature>

<Signature name="velocity" >
<OMOBJ xmlns="http://www.openmath.org/OpenMath">
  <OMS name="MonoidDimension"/>
</OMOBJ>
</Signature>

<Signature name="power" >
<OMOBJ xmlns="http://www.openmath.org/OpenMath">
  <OMS name="MonoidDimension"/>
</OMOBJ>
</Signature>

<Signature name="acceleration" >
<OMOBJ xmlns="http://www.openmath.org/OpenMath">
  <OMS name="MonoidDimension"/>
</OMOBJ>
</Signature>

<Signature name="time" >
<OMOBJ xmlns="http://www.openmath.org/OpenMath">
  <OMS name="MonoidDimension"/>
</OMOBJ>
</Signature>

<Signature name="mass" >
<OMOBJ xmlns="http://www.openmath.org/OpenMath">
  <OMS name="MonoidDimension"/>
</OMOBJ>
</Signature>

<Signature name="force" >
<OMOBJ xmlns="http://www.openmath.org/OpenMath">
  <OMS name="MonoidDimension"/>
</OMOBJ>
</Signature>

<Signature name="temperature" >
```

```
<OMOBJ xmlns="http://www.openmath.org/OpenMath">
  <OMS name="MonoidDimension"/>
</OMOBJ>
</Signature>

<Signature name="relativeTemperature" >
<OMOBJ xmlns="http://www.openmath.org/OpenMath">
  <OMS name="NonMonoidDimension"/>
</OMOBJ>
</Signature>

<Signature name="pressure" >
<OMOBJ xmlns="http://www.openmath.org/OpenMath">
  <OMS name="MonoidDimension"/>
</OMOBJ>
</Signature>

<Signature name="charge" >
<OMOBJ xmlns="http://www.openmath.org/OpenMath">
  <OMS name="MonoidDimension"/>
</OMOBJ>
</Signature>

<Signature name="current" >
<OMOBJ xmlns="http://www.openmath.org/OpenMath">
  <OMS name="MonoidDimension"/>
</OMOBJ>
</Signature>

<Signature name="voltage" >
<OMOBJ xmlns="http://www.openmath.org/OpenMath">
  <OMS name="MonoidDimension"/>
</OMOBJ>
</Signature>

<Signature name="resistance" >
<OMOBJ xmlns="http://www.openmath.org/OpenMath">
  <OMS name="MonoidDimension"/>
</OMOBJ>
</Signature>

<Signature name="density" >
<OMOBJ xmlns="http://www.openmath.org/OpenMath">
  <OMS name="MonoidDimension"/>
</OMOBJ>
</Signature>

<Signature name="energy" >
<OMOBJ xmlns="http://www.openmath.org/OpenMath">
  <OMS name="MonoidDimension"/>
</OMOBJ>
</Signature>

<Signature name="concentration" >
<OMOBJ xmlns="http://www.openmath.org/OpenMath">
  <OMS name="MonoidDimension"/>
</OMOBJ>
</Signature>

<Signature name="momentum" >
<OMOBJ xmlns="http://www.openmath.org/OpenMath">
  <OMS name="MonoidDimension"/>
</OMOBJ>
</Signature>

<Signature name="speed" >
<OMOBJ xmlns="http://www.openmath.org/OpenMath">
  <OMS name="MonoidDimension"/>
</OMOBJ>
</Signature>


<Signature name="displacement" >
<OMOBJ xmlns="http://www.openmath.org/OpenMath">
  <OMS name="MonoidDimension"/>
</OMOBJ>
</Signature>
</CDSignatures>
```

C.1.3 File: units_imperial1.ocd

```
<CD xmlns="http://www.openmath.org/OpenMathCD">
<CDName> units_imperial1 </CDName>
<CDURL> http://www.openmath.org/cd/units_imperial1.ocd
    </CDURL>
<CDReviewDate> 2008-03-31 </CDReviewDate>
<CDStatus> experimental </CDStatus>
<CDDate> 2004-08-27 </CDDate>
<CDVersion> 4 </CDVersion>
<CDRevision> 0 </CDRevision>

<Description>
This CD defines symbols to represent imperial standard
    measures.
</Description>

<CDDefinition>
<Name> foot </Name>
<Description>
This symbol represents the measure of one foot. This is
    the standard
imperial measure for distance.
</Description>
<CMP> 1 foot = 0.3048 metres </CMP>

<FMP><OMOBJ xmlns="http://www.openmath.org/OpenMath">
  <OMA>
    <OMS name="eq" cd="relation1"/>
    <OMA>
      <OMS name="times" cd="arith1"/>
      <OMI> 1 </OMI>
      <OMS name="foot" cd="units_imperial1"/>
    </OMA>
    <OMA>
      <OMS name="times" cd="arith1"/>
      <OMA>
        <OMS name="divide" cd="arith1"/>
        <OMI>3048</OMI>
        <OMI>10000</OMI>
      </OMA>
      <OMS name="metre" cd="units_metric1"/>
    </OMA>
  </OMA>
</OMOBJ></FMP>

</CDDefinition>

<CDDefinition>
<Name> yard </Name>
<Description>
This symbol represents the measure of one yard. This is a
standard imperial measure for distance, defined in terms
    of the foot.
</Description>
<CMP> 1 yard = 3 feet </CMP>

<FMP><OMOBJ xmlns="http://www.openmath.org/OpenMath">
  <OMA>
    <OMS name="eq" cd="relation1"/>
    <OMA>
      <OMS name="times" cd="arith1"/>
      <OMI> 1 </OMI>
      <OMS name="yard" cd="units_imperial1"/>
    </OMA>
    <OMA>
      <OMS name="times" cd="arith1"/>
      <OMI> 3 </OMI>
      <OMS name="foot" cd="units_imperial1"/>
    </OMA>
  </OMA>
</OMOBJ></FMP>

</CDDefinition>

<CDDefinition>
<Name> mile </Name>
<Description>
This symbol represents the measure of one (land, or
    statute) mile. This is a
standard imperial measure for distance, defined in terms
    of the foot.
</Description>
<CMP> 1 mile = 5280 feet </CMP>

<FMP><OMOBJ xmlns="http://www.openmath.org/OpenMath">
  <OMA>
    <OMS name="eq" cd="relation1"/>
    <OMA>
```

```
      <OMS name="times" cd="arith1"/>
      <OMI> 1 </OMI>
      <OMS name="mile" cd="units_imperial1"/>
    </OMA>
    <OMA>
      <OMS name="times" cd="arith1"/>
      <OMI> 5280 </OMI>
      <OMS name="foot" cd="units_imperial1"/>
    </OMA>
  </OMA>
</OMOBJ></FMP>

</CDDefinition>

<CDDefinition>
<Name> acre </Name>
<Description>
This symbol represents the measure of one acre. This is a standard
imperial measure for area.
</Description>
<CMP> 1 acre = 4840 square yards </CMP>

<FMP><OMOBJ xmlns="http://www.openmath.org/OpenMath">
  <OMA>
    <OMS name="eq" cd="relation1"/>
    <OMA>
      <OMS name="times" cd="arith1"/>
      <OMI> 1 </OMI>
      <OMS name="acre" cd="units_imperial1"/>
    </OMA>
    <OMA>
      <OMS name="times" cd="arith1"/>
      <OMI> 4840 </OMI>
      <OMA>
        <OMS name="times" cd="arith1"/>
        <OMS name="yard" cd="units_imperial1"/>
        <OMS name="yard" cd="units_imperial1"/>
      </OMA>
    </OMA>
  </OMA>
</OMOBJ></FMP>

</CDDefinition>

<CDDefinition>
<Name> pint </Name>
<Description>
This symbol represents the measure of one (imperial) pint. This is the standard
imperial measure for volume. See units_us1 for the U.S. pint.
</Description>

<CMP> 1 pint = 1/8 gallon </CMP>

<FMP><OMOBJ xmlns="http://www.openmath.org/OpenMath">
  <OMA>
    <OMS name="eq" cd="relation1"/>
    <OMA>
      <OMS name="times" cd="arith1"/>
      <OMI> 1 </OMI>
      <OMS name="pint" cd="units_imperial1"/>
    </OMA>
    <OMA>
      <OMS name="times" cd="arith1"/>
      <OMA>
        <OMS name="divide" cd="arith1"/>
        <OMI>1</OMI>
        <OMI>8</OMI>
      </OMA>
      <OMS name="gallon" cd="units_imperial1"/>
    </OMA>
  </OMA>
</OMOBJ></FMP>

</CDDefinition>

<CDDefinition>
<Name> pound_mass </Name>
<Description>
This symbol represents the measure of the mass which weighs one pound
under the influence of standard gravity.
</Description>

<CMP> 1 pound = 0.45359237 kilograms </CMP>

<FMP><OMOBJ xmlns="http://www.openmath.org/OpenMath">
  <OMA>
```

```
    <OMS name="eq" cd="relation1"/>
    <OMA>
      <OMS name="times" cd="arith1"/>
      <OMI> 1 </OMI>
      <OMS name="pound_mass" cd="units_imperial1"/>
    </OMA>
    <OMA>
      <OMS name="times" cd="arith1"/>
      <OMA>
        <OMS name="divide" cd="arith1"/>
        <OMI>45359237</OMI>
        <OMI>  100000</OMI>
      </OMA>
      <OMS name="gramme" cd="units_metric1"/>
    </OMA>
  </OMA>
</OMOBJ></FMP>

</CDDefinition>

<CDDefinition>
<Name> pound_force </Name>
<Description>
This symbol represents the measure of force of one pound.
</Description>

<CMP> 1 pound force = 4.44822 Newtons </CMP>

<FMP><OMOBJ xmlns="http://www.openmath.org/OpenMath">
  <OMA>
    <OMS name="eq" cd="relation1"/>
    <OMA>
      <OMS name="times" cd="arith1"/>
      <OMI> 1 </OMI>
      <OMS name="pound_force" cd="units_imperial1"/>
    </OMA>
    <OMA>
      <OMS name="times" cd="arith1"/>
      <OMA>
        <OMS name="divide" cd="arith1"/>
        <OMI>4448</OMI>
        <OMI>1000</OMI>
      </OMA>
      <OMS name="Newton" cd="units_metric1"/>
    </OMA>
  </OMA>
</OMOBJ></FMP>

</CDDefinition>

<CDDefinition>
<Name> degree_Fahrenheit </Name>
<Description>
This symbol represents the measure of one degree
    Fahrenheit. This is
the standard imperial measure for temperature.
</Description>

<CMP> 1 degree Fahrenheit = 5/9*(1-32) degrees Celsius
    </CMP>

<FMP><OMOBJ xmlns="http://www.openmath.org/OpenMath">
  <OMA>
    <OMS name="eq" cd="relation1"/>
    <OMA>
      <OMS name="times" cd="arith1"/>
      <OMI> 1 </OMI>
      <OMS name="degree_Fahrenheit"
          cd="units_imperial1"/>
    </OMA>
    <OMA>
      <OMS name="times" cd="arith1"/>
      <OMA>
        <OMS name="divide" cd="arith1"/>
        <OMI>5</OMI>
        <OMI>9</OMI>
      </OMA>
      <OMA>
        <OMS name="minus" cd="arith1"/>
        <OMS name="degree_Celsius" cd="units_metric1"/>
        <OMI> 32 </OMI>
      </OMA>
    </OMA>
  </OMA>
</OMOBJ></FMP>
</CDDefinition>

<CDDefinition>
<Name> relative_degree_Fahrenheit </Name>
<Description>
```

```
This symbol represents the measure of one relative degree Fahrenheit.
</Description>

<CMP> 1 relative degree Fahrenheit = 5/9 relative degrees Celsius </CMP>

<FMP><OMOBJ xmlns="http://www.openmath.org/OpenMath">
  <OMA>
    <OMS name="eq" cd="relation1"/>
    <OMA>
      <OMS name="times" cd="arith1"/>
      <OMI> 1 </OMI>
      <OMS name="relative_degree_Fahrenheit" cd="units_imperial1"/>
    </OMA>
    <OMA>
      <OMS name="times" cd="arith1"/>
      <OMA>
        <OMS name="divide" cd="arith1"/>
        <OMI>5</OMI>
        <OMI>9</OMI>
      </OMA>
      <OMS name="relative_degree_Celsius" cd="units_metric1"/>
    </OMA>
  </OMA>
</OMOBJ></FMP>
</CDDefinition>

<CDDefinition>
<Name> bar </Name>
<Description>
This symbol represents the measure of one bar. This is the standard
imperial measure for pressure.
</Description>

<CMP> 1 bar = 100 000 Pascals </CMP>

<FMP><OMOBJ xmlns="http://www.openmath.org/OpenMath">
  <OMA>
    <OMS name="eq" cd="relation1"/>
    <OMA>
      <OMS name="times" cd="arith1"/>
      <OMI> 1 </OMI>
      <OMS name="bar" cd="units_imperial1"/>
    </OMA>
    <OMA>
      <OMS name="times" cd="arith1"/>
      <OMI> 100000 </OMI>
      <OMS name="Pascal" cd="units_metric1"/>
    </OMA>
  </OMA>
</OMOBJ></FMP>
</CDDefinition>

<CDDefinition>
<Name> inch </Name>
<Description>
This symbol represents the measure of one rod.
</Description>
<CMP> 1 inch = 1/12 foot </CMP>

<FMP><OMOBJ xmlns="http://www.openmath.org/OpenMath">
  <OMA>
    <OMS name="eq" cd="relation1"/>
    <OMA>
      <OMS name="times" cd="arith1"/>
      <OMI> 1 </OMI>
      <OMS name="inch" cd="units_imperial1"/>
    </OMA>
    <OMA>
      <OMS name="times" cd="arith1"/>
      <OMA>
  <OMS name="divide" cd="arith1"/>
    <OMI> 1 </OMI>
  <OMI> 12 </OMI>
      </OMA>
      <OMS name="foot" cd="units_imperial1"/>
    </OMA>
  </OMA>
</OMOBJ></FMP>
</CDDefinition>

<CDDefinition>
<Name> mil </Name>
<Description>
This symbol represents the measure of one rod.
</Description>
```

```
<CMP> 1 mil = 1/1000 inch </CMP>

<FMP><OMOBJ xmlns="http://www.openmath.org/OpenMath">
  <OMA>
    <OMS name="eq" cd="relation1"/>
    <OMA>
      <OMS name="times" cd="arith1"/>
      <OMI> 1 </OMI>
      <OMS name="mil" cd="units_imperial1"/>
    </OMA>
    <OMA>
      <OMS name="times" cd="arith1"/>
      <OMA>
  <OMS name="divide" cd="arith1"/>
  <OMI> 1 </OMI>
  <OMI> 1000 </OMI>
      </OMA>
      <OMS name="inch" cd="units_imperial1"/>
    </OMA>
  </OMA>
</OMOBJ></FMP>
</CDDefinition>

<CDDefinition>
<Name> furlong </Name>
<Description>
This symbol represents the measure of one furlong.
</Description>
<CMP> 1 furlong = 220 yard </CMP>

<FMP><OMOBJ xmlns="http://www.openmath.org/OpenMath">
  <OMA>
    <OMS name="eq" cd="relation1"/>
    <OMA>
      <OMS name="times" cd="arith1"/>
      <OMI> 1 </OMI>
      <OMS name="furlong" cd="units_imperial1"/>
    </OMA>
    <OMA>
      <OMS name="times" cd="arith1"/>
      <OMI>220</OMI>
      <OMS name = "yard" cd="units_imperial1"/>
    </OMA>
  </OMA>
</OMOBJ></FMP>
</CDDefinition>

<CDDefinition>
<Name> stone </Name>
<Description>
This symbol represents the measure of one stone.
</Description>
<CMP> 1 stone = 14 pound </CMP>

<FMP><OMOBJ xmlns="http://www.openmath.org/OpenMath">
  <OMA>
    <OMS name="eq" cd="relation1"/>
    <OMA>
      <OMS name="times" cd="arith1"/>
      <OMI> 1 </OMI>
      <OMS name="stone" cd="units_imperial1"/>
    </OMA>
    <OMA>
      <OMS name="times" cd="arith1"/>
      <OMI>14</OMI>
      <OMS name = "pound_mass" cd="units_imperial1"/>
    </OMA>
  </OMA>
</OMOBJ></FMP>
</CDDefinition>

<CDDefinition>
<Name> ton_long </Name>
<Description>
This symbol represents the measure of one ton.
</Description>
<CMP> 1 ton = 2240 pound </CMP>

<FMP><OMOBJ xmlns="http://www.openmath.org/OpenMath">
  <OMA>
    <OMS name="eq" cd="relation1"/>
    <OMA>
      <OMS name="times" cd="arith1"/>
      <OMI> 1 </OMI>
      <OMS name="ton_long" cd="units_imperial1"/>
    </OMA>
    <OMA>
      <OMS name="times" cd="arith1"/>
      <OMI>2240</OMI>
      <OMS name = "pound_mass" cd="units_imperial1"/>
```

```
      </OMA>
  </OMA>
</OMOBJ></FMP>
</CDDefinition>

<CDDefinition>
<Name> ounce </Name>
<Description>
This symbol represents the measure of one ounce.
</Description>
<CMP> 1 ounce = 1/16 pound </CMP>

<FMP><OMOBJ xmlns="http://www.openmath.org/OpenMath">
  <OMA>
    <OMS name="eq" cd="relation1"/>
    <OMA>
      <OMS name="times" cd="arith1"/>
      <OMI> 1 </OMI>
      <OMS name="ounce" cd="units_imperial1"/>
    </OMA>
    <OMA>
      <OMS name="times" cd="arith1"/>
      <OMA>
        <OMS name="divide" cd="arith1" />
  <OMI>1</OMI>
        <OMI>16</OMI>
      </OMA>
      <OMS name = "pound_mass" cd="units_imperial1"/>
    </OMA>
  </OMA>
</OMOBJ></FMP>
</CDDefinition>
<CDDefinition>
<Name> gallon </Name>
<Description>
This symbol represents the measure of one gallon.
</Description>
<CMP> 1 gallon = 4.54609 litres </CMP>

<FMP><OMOBJ xmlns="http://www.openmath.org/OpenMath">
  <OMA>
    <OMS name="eq" cd="relation1"/>
    <OMA>
      <OMS name="times" cd="arith1"/>
      <OMI> 1 </OMI>
      <OMS name="gallon" cd="units_imperial1"/>
    </OMA>
    <OMA>
      <OMS name="times" cd="arith1"/>
      <OMA>
        <OMS name="divide" cd="arith1"/>
        <OMI>454609</OMI>
        <OMI>100000</OMI>
      </OMA>
      <OMS name = "litre" cd="units_metric1"/>
    </OMA>
  </OMA>
</OMOBJ></FMP>
</CDDefinition>
<CDDefinition>
<Name> fluid_ounce </Name>
<Description>
This symbol represents the measure of one fluid ounce.
</Description>
<CMP> 1 fluid ounce = 1/160 gallon </CMP>

<FMP><OMOBJ xmlns="http://www.openmath.org/OpenMath">
  <OMA>
    <OMS name="eq" cd="relation1"/>
    <OMA>
      <OMS name="times" cd="arith1"/>
      <OMI> 1 </OMI>
      <OMS name="fluid_ounce" cd="units_imperial1"/>
    </OMA>
    <OMA>
      <OMS name="times" cd="arith1"/>
      <OMA>
        <OMS name="divide" cd="arith1" />
        <OMI>1</OMI>
        <OMI>160</OMI>
      </OMA>
      <OMS name = "gallon" cd="units_imperial1"/>
    </OMA>
  </OMA>
</OMOBJ></FMP>
</CDDefinition>
<CDDefinition>
<Name> acre_foot </Name>
<Description>
This symbol represents the measure of one gallon.
```

```
</Description>
<CMP> 1 acre_foot = 1 acre * 1 foot = 43560 foot_cubed
    </CMP>

<FMP><OMOBJ xmlns="http://www.openmath.org/OpenMath">
  <OMA>
    <OMS name="eq" cd="relation1"/>
    <OMA>
      <OMS name="times" cd="arith1"/>
      <OMI> 1 </OMI>
      <OMS name="acre_foot" cd="units_imperial1"/>
    </OMA>
    <OMA>
      <OMS name="times" cd="arith1"/>
      <OMI>1</OMI>
      <OMA>
        <OMS name="times" cd="arith1"/>
        <OMI> 43560 </OMI>
        <OMA>
          <OMS name="power" cd="arith1"/>
          <OMS name = "foot" cd="units_imperial1"/>
          <OMI>3</OMI>
        </OMA>
      </OMA>
    </OMA>
  </OMA>
</OMOBJ></FMP>
</CDDefinition>
</CD>
```

C.1.4 File: units_imperial1.sts

```
<CDSignatures
    xmlns="http://www.openmath.org/OpenMathCDS"
    type="sts" cd="units_imperial1">
<CDSStatus> experimental </CDSStatus>

<Signature name="foot" >
<OMOBJ xmlns="http://www.openmath.org/OpenMath">
  <OMS cd="dimensions1" name="length"/>
</OMOBJ>
</Signature>

<Signature name="yard" >
<OMOBJ xmlns="http://www.openmath.org/OpenMath">
  <OMS cd="dimensions1" name="length"/>
</OMOBJ>
</Signature>

<Signature name="mile" >
<OMOBJ xmlns="http://www.openmath.org/OpenMath">
  <OMS cd="dimensions1" name="length"/>
</OMOBJ>
</Signature>

<Signature name="acre" >
<OMOBJ xmlns="http://www.openmath.org/OpenMath">
  <OMS cd="dimensions1" name="area"/>
</OMOBJ>
</Signature>

<Signature name="pint" >
<OMOBJ xmlns="http://www.openmath.org/OpenMath">
  <OMS cd="dimensions1" name="volume"/>
</OMOBJ>
</Signature>

<Signature name="pound_mass" >
<OMOBJ xmlns="http://www.openmath.org/OpenMath">
  <OMS cd="dimensions1" name="mass"/>
</OMOBJ>
</Signature>

<Signature name="pound_force" >
<OMOBJ xmlns="http://www.openmath.org/OpenMath">
  <OMS cd="dimensions1" name="force"/>
</OMOBJ>
</Signature>

<Signature name="degree_Fahrenheit" >
<OMOBJ xmlns="http://www.openmath.org/OpenMath">
  <OMS cd="dimensions1" name="temperature"/>
</OMOBJ>
</Signature>
```

```
<Signature name="relative_degree_Fahrenheit" >
<OMOBJ xmlns="http://www.openmath.org/OpenMath">
  <OMS cd="dimensions1" name="relativeTemperature"/>
</OMOBJ>
</Signature>

<Signature name="bar" >
<OMOBJ xmlns="http://www.openmath.org/OpenMath">
  <OMS cd="dimensions1" name="pressure"/>
</OMOBJ>
</Signature>
<Signature name="inch" >
<OMOBJ xmlns="http://www.openmath.org/OpenMath">
  <OMS cd="dimensions1" name="length"/>
</OMOBJ>
</Signature>
<Signature name="mil" >
<OMOBJ xmlns="http://www.openmath.org/OpenMath">
  <OMS cd="dimensions1" name="length"/>
</OMOBJ>
</Signature>
<Signature name="furlong" >
<OMOBJ xmlns="http://www.openmath.org/OpenMath">
  <OMS cd="dimensions1" name="length"/>
</OMOBJ>
</Signature>
<Signature name="stone" >
<OMOBJ xmlns="http://www.openmath.org/OpenMath">
  <OMS cd="dimensions1" name="mass"/>
</OMOBJ>
</Signature>
<Signature name="ton_long" >
<OMOBJ xmlns="http://www.openmath.org/OpenMath">
  <OMS cd="dimensions1" name="mass"/>
</OMOBJ>
</Signature>
<Signature name="ounce" >
<OMOBJ xmlns="http://www.openmath.org/OpenMath">
  <OMS cd="dimensions1" name="mass"/>
</OMOBJ>
</Signature>
<Signature name="gallon" >
<OMOBJ xmlns="http://www.openmath.org/OpenMath">
  <OMS cd="dimensions1" name="volume"/>
</OMOBJ>
</Signature>
<Signature name="fluid_ounce" >
<OMOBJ xmlns="http://www.openmath.org/OpenMath">
  <OMS cd="dimensions1" name="volume"/>
</OMOBJ>
</Signature>
<Signature name="acre_foot" >
<OMOBJ xmlns="http://www.openmath.org/OpenMath">
  <OMS cd="dimensions1" name="volume"/>
</OMOBJ>
</Signature>

</CDSignatures>
```

C.1.5 File: unit_metric1.ocd

```
<CD xmlns="http://www.openmath.org/OpenMathCD">
<CDName> units_metric1 </CDName>
<CDURL> http://www.openmath.org/cd/units_metric1.ocd
    </CDURL>
<CDReviewDate> 2008-03-31 </CDReviewDate>
<CDStatus> experimental </CDStatus>
<CDDate> 2004-08-27 </CDDate>
<CDVersion> 4 </CDVersion>
<CDRevision> 0 </CDRevision>

<Description>
This CD defines symbols to represent the basic physical
    units in the SI
(syst\'eme international) system of units. It should
    probably be renamed
units_si.
</Description>

<CDDefinition>
<Name> metre </Name>
<Description>
```

```
This symbol represents the measure of one metre. This is the standard
SI unit measure for physical distance.
</Description>
<CMP> This is a base unit for the SI system </CMP>
</CDDefinition>

<CDDefinition>
<Name> litre </Name>
<Description>
This symbol represents the measure of one litre. This is a standard
metric measure for physical volume.
</Description>
<CMP> A litre is, since 1964, a cubic decimetre, or a thousandth of a cubic
      metre, as the FMP below states. </CMP>

<FMP><OMOBJ xmlns="http://www.openmath.org/OpenMath">
<OMA>
  <OMS name="eq" cd="relation1"/>
  <OMA>
    <OMS name="times" cd="arith1"/>
    <OMI> 1 </OMI>
    <OMS name="litre" cd="units_metric1"/>
  </OMA>
  <OMA>
    <OMS name="times" cd="arith1"/>
    <OMA>
      <OMS name="divide" cd="arith1"/>
      <OMI>1</OMI>
      <OMI>1000</OMI>
    </OMA>
    <OMA>
      <OMS name="power" cd="arith1"/>
      <OMS name="metre" cd="units_metric1"/>
      <OMI> 3 </OMI>
    </OMA>
  </OMA>
</OMA>
</OMOBJ></FMP>

</CDDefinition>

<CDDefinition>
<Name> second </Name>
<Description>
This symbol represents the measure of one second. This is the standard
SI measure for time.
</Description>
<CMP> The Si unit is the same as the UTC unit, to which we refer </CMP>

<FMP><OMOBJ xmlns="http://www.openmath.org/OpenMath">
<OMA>
  <OMS name="eq" cd="relation1"/>
  <OMS name="second" cd="units_metric1"/>
  <OMS name="second" cd="units_time1"/>
</OMA>
</OMOBJ></FMP>

</CDDefinition>

<CDDefinition>
<Name> gramme </Name>
<Description>
  This symbol represents the measure of one gramme. This is not quite the
  standard SI measure for mass, which is the kilogramme, but OpenMath
  chooses to regard the gramme as standard, otherwise one would have to call
  it the milli-kilogramme.
</Description>

<CMP> This is a basic unit of the SI system </CMP>

</CDDefinition>

<CDDefinition>
<Name> Newton </Name>
<Description>
This symbol represents the measure of one Newton. This is the standard
SI measure for force.
</Description>
</CDDefinition>

<CDDefinition>
```

```
<Name> Joule </Name>
<Description>
This symbol represents the measure of one Joule. This is the standard
SI measure for energy.
</Description>
</CDDefinition>

<CDDefinition>
<Name> Watt </Name>
<Description>
This symbol represents the measure of one Watt. This is the standard
SI measure for power.
</Description>
<FMP><OMOBJ xmlns="http://www.openmath.org/OpenMath">
  <OMA>
    <OMS name="eq" cd="relation1"/>
    <OMA>
      <OMS name="times" cd="arith1"/>
      <OMI> 1 </OMI>
      <OMS name="Watt" cd="units_metric1"/>
    </OMA>
    <OMA>
      <OMS name="times" cd="arith1"/>
      <OMI> 1 </OMI>
      <OMA>
        <OMS name="divide" cd="arith1"/>
        <OMA>
          <OMS name="times" cd="arith1"/>
          <OMI>1</OMI>
          <OMS name="Joule" cd="units_metric1"/>
        </OMA>
        <OMA>
          <OMS name="times" cd="arith1"/>
          <OMI>1</OMI>
          <OMS name="second" cd="units_time1"/>
        </OMA>
      </OMA>
    </OMA>
  </OMA>
</OMOBJ></FMP>
</CDDefinition>

<CDDefinition>
<Name> degree_Kelvin </Name>
<Description>
This symbol represents the measure of one degree Kelvin. This is a standard
SI measure for temperature relative to absolute zero.
</Description>

<CMP> 1 degree Kelvin = 1 - 273.15 degrees Celsius </CMP>

<FMP><OMOBJ xmlns="http://www.openmath.org/OpenMath">
  <OMA>
    <OMS name="eq" cd="relation1"/>
    <OMA>
      <OMS name="times" cd="arith1"/>
      <OMI> 1 </OMI>
      <OMS name="degree_Kelvin" cd="units_metric1"/>
    </OMA>
    <OMA>
      <OMS name="minus" cd="arith1"/>
      <OMS name="degree_Celsius" cd="units_metric1"/>
      <OMA>
        <OMS name="divide" cd="arith1"/>
        <OMI>27315</OMI>
        <OMI>100</OMI>
      </OMA>
    </OMA>
  </OMA>
</OMOBJ></FMP>

</CDDefinition>

<CDDefinition>
<Name> degree_Celsius </Name>
<Description>
This symbol represents the measure of one degree Celsius. This is a standard
metric measure for temperature.
</Description>
</CDDefinition>

<CDDefinition>
<Name> relative_degree_Kelvin </Name>
<Description>
This symbol represents the measure of one relative degree Kelvin.
```

```
</Description>

<CMP> 1 relative degree Kelvin = 1 relative degree
    Celsius </CMP>

<FMP><OMOBJ xmlns="http://www.openmath.org/OpenMath">
  <OMA>
    <OMS name="eq" cd="relation1"/>
    <OMA>
      <OMS name="times" cd="arith1"/>
      <OMI> 1 </OMI>
      <OMS name="relative_degree_Kelvin"
          cd="units_metric1"/>
    </OMA>
    <OMA>
      <OMS name="times" cd="arith1"/>
      <OMI> 1 </OMI>
      <OMS name="relative_degree_Celsius"
          cd="units_metric1"/>
    </OMA>
  </OMA>
</OMOBJ></FMP>

</CDDefinition>

<CDDefinition>
<Name> relative_degree_Celsius </Name>
<Description>
This symbol represents the measure of one relative
    degree Celsius.
</Description>
</CDDefinition>

<CDDefinition>
<Name> Pascal </Name>
<Description>
This symbol represents the measure of one Newton per
    square metre.
This is the standard SI measure for pressure.
</Description>
<FMP>
<OMOBJ xmlns="http://www.openmath.org/OpenMath">
  <OMA>
    <OMS name="eq" cd="relation1"/>
    <OMA>
      <OMS name="times" cd="arith1"/>
      <OMI>1</OMI>
      <OMS name="Pascal" cd="units_metric1"/>
    </OMA>
    <OMA>
      <OMS name="times" cd="arith1"/>
      <OMI>1</OMI>
      <OMA>
        <OMS name="divide" cd="arith1"/>
        <OMS name="Newton" cd="units_metric1"/>
        <OMA>
          <OMS name="times" cd="arith1"/>
          <OMS name="metre" cd="units_metric1"/>
          <OMS name="metre" cd="units_metric1"/>
        </OMA>
      </OMA>
    </OMA>
  </OMA>
</OMOBJ>
</FMP>
</CDDefinition>

<CDDefinition>
<Name> Coulomb </Name>
<Description>
This symbol represents the measure of one Coulomb. This
    is the standard
SI measure for charge.
</Description>
</CDDefinition>

<CDDefinition>
<Name> amp </Name>
<Description>
This symbol represents the measure of one amp. This is
    the standard
SI measure for current.
</Description>
</CDDefinition>

<CDDefinition>
<Name> volt </Name>
<Description>
This symbol represents the measure of one volt. This is
    the standard
```

```
SI measure for voltage.
</Description>
</CDDefinition>
```

C.1.6 File: units_metric1.sts

```
<CDSignatures
    xmlns="http://www.openmath.org/OpenMathCDS"
    type="sts" cd="units_metric1">
<CDSStatus> experimental </CDSStatus>

<Signature name="metre" >
<OMOBJ xmlns="http://www.openmath.org/OpenMath">
  <OMS cd="dimensions1" name="length"/>
</OMOBJ>
</Signature>

<Signature name="litre" >
<OMOBJ xmlns="http://www.openmath.org/OpenMath">
  <OMS cd="dimensions1" name="volume"/>
</OMOBJ>
</Signature>

<Signature name="second" >
<OMOBJ xmlns="http://www.openmath.org/OpenMath">
  <OMS cd="dimensions1" name="time"/>
</OMOBJ>
</Signature>

<Signature name="gramme" >
<OMOBJ xmlns="http://www.openmath.org/OpenMath">
  <OMS cd="dimensions1" name="mass"/>
</OMOBJ>
</Signature>

<Signature name="Newton" >
<OMOBJ xmlns="http://www.openmath.org/OpenMath">
  <OMS cd="dimensions1" name="force"/>
</OMOBJ>
</Signature>

<Signature name="Joule" >
<OMOBJ xmlns="http://www.openmath.org/OpenMath">
  <OMS cd="dimensions1" name="energy"/>
</OMOBJ>
</Signature>

<Signature name="Watt" >
<OMOBJ xmlns="http://www.openmath.org/OpenMath">
  <OMS cd="dimensions1" name="power"/>
</OMOBJ>
</Signature>

<Signature name="degree_Kelvin" >
<OMOBJ xmlns="http://www.openmath.org/OpenMath">
  <OMS cd="dimensions1" name="temperature"/>
</OMOBJ>
</Signature>

<Signature name="degree_Celsius" >
<OMOBJ xmlns="http://www.openmath.org/OpenMath">
  <OMS cd="dimensions1" name="temperature"/>
</OMOBJ>
</Signature>

<Signature name="relative_degree_Kelvin" >
<OMOBJ xmlns="http://www.openmath.org/OpenMath">
  <OMS cd="dimensions1" name="relativeTemperature"/>
</OMOBJ>
</Signature>

<Signature name="relative_degree_Celsius" >
<OMOBJ xmlns="http://www.openmath.org/OpenMath">
  <OMS cd="dimensions1" name="relativeTemperature"/>
</OMOBJ>
</Signature>

<Signature name="Pascal" >
<OMOBJ xmlns="http://www.openmath.org/OpenMath">
  <OMS cd="dimensions1" name="pressure"/>
```

```
</CD>
```

```
</OMOBJ>
</Signature>

<Signature name="Coulomb" >
<OMOBJ xmlns="http://www.openmath.org/OpenMath">
  <OMS cd="dimensions1" name="charge"/>
</OMOBJ>
</Signature>

<Signature name="amp" >
<OMOBJ xmlns="http://www.openmath.org/OpenMath">
  <OMS cd="dimensions1" name="current"/>
</OMOBJ>
</Signature>

<Signature name="volt" >
<OMOBJ xmlns="http://www.openmath.org/OpenMath">
  <OMS cd="dimensions1" name="voltage"/>
</OMOBJ>
</Signature>

</CDSignatures>
```

C.1.7 File: units_us1.ocd

```
<CD xmlns="http://www.openmath.org/OpenMathCD">
<CDName> units_us1 </CDName>
<CDURL> http://www.openmath.org/cd/units_us1.ocd </CDURL>
<CDReviewDate> 2008-03-31 </CDReviewDate>
<CDStatus> experimental </CDStatus>
<CDDate> 2004-08-27 </CDDate>
<CDVersion> 4 </CDVersion>
<CDRevision> 0 </CDRevision>


<Description>
This CD defines symbols to represent U.S. customary unit
    measures.
</Description>

<CDDefinition>
<Name> foot_us_survey </Name>
<Description>
This symbol represents the measure of one U.S. Survey
    foot.
</Description>
<CMP> 1 U.S. Survey foot = 1200/3937 metres </CMP>

<FMP><OMOBJ xmlns="http://www.openmath.org/OpenMath">
  <OMA>
    <OMS name="eq" cd="relation1"/>
    <OMA>
      <OMS name="times" cd="arith1"/>
      <OMI> 1 </OMI>
      <OMS name="foot_us_survey" cd="units_us1"/>
    </OMA>
    <OMA>
      <OMS name="times" cd="arith1"/>
      <OMA>
        <OMS name="divide" cd="arith1"/>
        <OMI> 1200 </OMI>
        <OMI> 3937 </OMI>
      </OMA>
      <OMS name="metre" cd="units_metric1"/>
    </OMA>
  </OMA>
</OMOBJ></FMP>
</CDDefinition>

<CDDefinition>
<Name> yard_us_survey </Name>
<Description>
This symbol represents the measure of one U.S. Survey
    yard.
</Description>
<CMP> 1 U.S. Survey yard = 3 U.S. Survey feet </CMP>

<FMP><OMOBJ xmlns="http://www.openmath.org/OpenMath">
  <OMA>
    <OMS name="eq" cd="relation1"/>
    <OMA>
      <OMS name="times" cd="arith1"/>
      <OMI> 1 </OMI>
```

```
      <OMS name="yard_us_survey" cd="units_us1"/>
    </OMA>
    <OMA>
      <OMS name="times" cd="arith1"/>
      <OMI> 3 </OMI>
      <OMS name="foot_us_survey" cd="units_us1"/>
    </OMA>
  </OMA>
</OMOBJ></FMP>
</CDDefinition>

<CDDefinition>
<Name> mile_us_survey </Name>
<Description>
This symbol represents the measure of one U.S. Survey
    mile.
</Description>
<CMP> 1 U.S. Survey mile = 5280 U.S. Survey feet </CMP>

<FMP><OMOBJ xmlns="http://www.openmath.org/OpenMath">
  <OMA>
    <OMS name="eq" cd="relation1"/>
    <OMA>
      <OMS name="times" cd="arith1"/>
      <OMI> 1 </OMI>
      <OMS name="mile_us_survey" cd="units_us1"/>
    </OMA>
    <OMA>
      <OMS name="times" cd="arith1"/>
      <OMI> 5280 </OMI>
      <OMS name="foot_us_survey" cd="units_us1"/>
    </OMA>
  </OMA>
</OMOBJ></FMP>
</CDDefinition>

<CDDefinition>
<Name> acre_us_survey </Name>
<Description>
This symbol represents the measure of one U.S. Survey
    acre.
</Description>
<CMP> 1 U.S. Survey acre = 4840 square U.S. Survey yards
    </CMP>

<FMP><OMOBJ xmlns="http://www.openmath.org/OpenMath">
  <OMA>
    <OMS name="eq" cd="relation1"/>
    <OMA>
      <OMS name="times" cd="arith1"/>
      <OMI> 1 </OMI>
      <OMS name="acre_us_survey" cd="units_us1"/>
    </OMA>
    <OMA>
      <OMS name="times" cd="arith1"/>
      <OMI> 4840 </OMI>
      <OMA>
        <OMS name="times" cd="arith1"/>
        <OMS name="yard_us_survey" cd="units_us1"/>
        <OMS name="yard_us_survey" cd="units_us1"/>
      </OMA>
    </OMA>
  </OMA>
</OMOBJ></FMP>
</CDDefinition>

<CDDefinition>
<Name> pint_us_dry </Name>
<Description>
This symbol represents the measure of one U.S. dry pint.
</Description>

<CMP> 1 U.S. dry pint = 0.5506104713575 litres </CMP>

<FMP><OMOBJ xmlns="http://www.openmath.org/OpenMath">
  <OMA>
    <OMS name="eq" cd="relation1"/>
    <OMA>
      <OMS name="times" cd="arith1"/>
      <OMI> 1 </OMI>
      <OMS name="pint_us_dry" cd="units_us1"/>
    </OMA>
    <OMA>
      <OMS name="times" cd="arith1"/>
      <OMA>
        <OMS name="divide" cd="arith1"/>
        <OMI>  5506104713575 </OMI>
        <OMI> 10000000000000 </OMI>
      </OMA>
      <OMS name="litre" cd="units_metric1"/>
```

```
      </OMA>
144   </OMA>
  </OMOBJ></FMP>
146 </CDDefinition>

148 <CDDefinition>
  <Name> pint_us_liquid </Name>
150 <Description>
  This symbol represents the measure of one U.S. liquid
      pint.
152 </Description>

154 <CMP> 1 U.S. liquid pint = 16 U.S. fluid ounces </CMP>

156 <FMP><OMOBJ xmlns="http://www.openmath.org/OpenMath">
    <OMA>
158   <OMS name="eq" cd="relation1"/>
      <OMA>
160     <OMS name="times" cd="arith1"/>
        <OMI> 1 </OMI>
162     <OMS name="pint_us_liquid" cd="units_us1"/>
      </OMA>
164   <OMA>
        <OMS name="times" cd="arith1"/>
166     <OMI> 16 </OMI>
        <OMS name="fluid_ounce_us" cd="units_us1"/>
168   </OMA>
    </OMA>
170 </OMOBJ></FMP>
  </CDDefinition>
172 <CDDefinition>
  <Name> fluid_ounce_us </Name>
174 <Description>
  This symbol represents the measure of one U.S. fluid
      ounce.
176 </Description>

178 <CMP> 1 U.S. fluid ounce = 29.573 5295625 millilitres
      </CMP>

180 <FMP><OMOBJ xmlns="http://www.openmath.org/OpenMath">
    <OMA>
182   <OMS name="eq" cd="relation1"/>
      <OMA>
184     <OMS name="times" cd="arith1"/>
        <OMI> 1 </OMI>
186     <OMS name="fluid_ounce_us" cd="units_us1"/>
      </OMA>
188   <OMA>
        <OMS name="times" cd="arith1"/>
190     <OMA>
          <OMS name="divide" cd="arith1"/>
192       <OMI>   295735295625 </OMI>
          <OMI> 10000000000000 </OMI>
194     </OMA>
        <OMS name="litre" cd="units_metric1"/>
196   </OMA>
    </OMA>
198 </OMOBJ></FMP>
  </CDDefinition>
200 <CDDefinition>
  <Name> cup_us </Name>
202 <Description>
  This symbol represents the measure of one U.S. cup.
204 </Description>

206 <CMP> 1 U.S. cup = 8 U.S. fluid ounce </CMP>

208 <FMP><OMOBJ xmlns="http://www.openmath.org/OpenMath">
    <OMA>
210   <OMS name="eq" cd="relation1"/>
      <OMA>
212     <OMS name="times" cd="arith1"/>
        <OMI> 1 </OMI>
214     <OMS name="cup_us" cd="units_us1"/>
      </OMA>
216   <OMA>
        <OMS name="times" cd="arith1"/>
218     <OMI> 8 </OMI>
        <OMS name="fluid_ounce_us" cd="units_us1"/>
220   </OMA>
    </OMA>
222 </OMOBJ></FMP>
  </CDDefinition>
224 <CDDefinition>
  <Name> gallon_us_liquid </Name>
226 <Description>
  This symbol represents the measure of one U.S. liquid
      gallon
228 </Description>
```

```
<CMP> 1 U.S. liquid gallon = 8 U.S. liquid pints </CMP>

<FMP><OMOBJ xmlns="http://www.openmath.org/OpenMath">
  <OMA>
    <OMS name="eq" cd="relation1"/>
    <OMA>
      <OMS name="times" cd="arith1"/>
      <OMI> 1 </OMI>
      <OMS name="gallon_us_liquid" cd="units_us1"/>
    </OMA>
    <OMA>
      <OMS name="times" cd="arith1"/>
      <OMI> 8 </OMI>
      <OMS name="pint_us_liquid" cd="units_us1"/>
    </OMA>
  </OMA>
</OMOBJ></FMP>
</CDDefinition>
<CDDefinition>
<Name> gallon_us_dry </Name>
<Description>
  This symbol represents the measure of one U.S. dry gallon
</Description>

<CMP> 1 U.S. dry gallon = 8 U.S. dry pints </CMP>

<FMP><OMOBJ xmlns="http://www.openmath.org/OpenMath">
  <OMA>
    <OMS name="eq" cd="relation1"/>
    <OMA>
      <OMS name="times" cd="arith1"/>
      <OMI> 1 </OMI>
      <OMS name="gallon_us_dry" cd="units_us1"/>
    </OMA>
    <OMA>
      <OMS name="times" cd="arith1"/>
      <OMI> 8 </OMI>
      <OMS name="pint_us_dry" cd="units_us1"/>
    </OMA>
  </OMA>
</OMOBJ></FMP>
</CDDefinition>
</CD>
```

C.1.8 File: units_us1.sts

```
<CDSignatures
    xmlns="http://www.openmath.org/OpenMathCDS"
    type="sts" cd="units_us1">
<CDSStatus> experimental </CDSStatus>

<Signature name="foot_us_survey" >
<OMOBJ xmlns="http://www.openmath.org/OpenMath">
  <OMS cd="dimensions1" name="length"/>
</OMOBJ>
</Signature>

<Signature name="yard_us_survey" >
<OMOBJ xmlns="http://www.openmath.org/OpenMath">
  <OMS cd="dimensions1" name="length"/>
</OMOBJ>
</Signature>

<Signature name="mile_us_survey" >
<OMOBJ xmlns="http://www.openmath.org/OpenMath">
  <OMS cd="dimensions1" name="length"/>
</OMOBJ>
</Signature>

<Signature name="acre_us_survey" >
<OMOBJ xmlns="http://www.openmath.org/OpenMath">
  <OMS cd="dimensions1" name="area"/>
</OMOBJ>
</Signature>

<Signature name="pint_us_dry" >
<OMOBJ xmlns="http://www.openmath.org/OpenMath">
  <OMS cd="dimensions1" name="volume"/>
</OMOBJ>
</Signature>

<Signature name="pint_us_liquid" >
```

```
<OMOBJ xmlns="http://www.openmath.org/OpenMath">
  <OMS cd="dimensions1" name="volume"/>
</OMOBJ>
</Signature>

<Signature name="fluid_ounce_us" >
<OMOBJ xmlns="http://www.openmath.org/OpenMath">
  <OMS cd="dimensions1" name="volume"/>
</OMOBJ>
</Signature>

<Signature name="cup_us" >
<OMOBJ xmlns="http://www.openmath.org/OpenMath">
  <OMS cd="dimensions1" name="volume"/>
</OMOBJ>
</Signature>

<Signature name="gallon_us_liquid" >
<OMOBJ xmlns="http://www.openmath.org/OpenMath">
  <OMS cd="dimensions1" name="volume"/>
</OMOBJ>
</Signature>

<Signature name="gallon_us_dry" >
<OMOBJ xmlns="http://www.openmath.org/OpenMath">
  <OMS cd="dimensions1" name="volume"/>
</OMOBJ>
</Signature>
</CDSignatures>
```

C.2 New CDs

C.2.1 File: units_metric_obsolete1.ocd

```
<CD xmlns="http://www.openmath.org/OpenMathCD">
<CDName> units_metric_obsolete1 </CDName>
<CDURL>
    http://www.openmath.org/cd/units_metric_obselote1.ocd
    </CDURL>
<CDReviewDate> 2008-03-31 </CDReviewDate>
<CDStatus> experimental </CDStatus>
<CDDate> 2008-03-31 </CDDate>
<CDVersion> 1 </CDVersion>
<CDRevision> 0 </CDRevision>

<Description>
This CD defines symbols to represent obsolete metric
    standard measures.
</Description>

<CDDefinition>
<Name> litre_pre1964 </Name>
<Description>
This symbol represents the previous (1901-1964) measure
    of one litre. This
used to be a standard metric measure for physical volume.
</Description>
<CMP> A litre is, since 1901 and until 1964, the volume
    occupied by a
    kilogramme of water at maximum density and
        standard pressure.
    The difference is about 0.0028%. </CMP>

<FMP><OMOBJ xmlns="http://www.openmath.org/OpenMath">
  <OMA>
    <OMS name="eq" cd="relation1"/>
    <OMA>
      <OMS name="times" cd="arith1"/>
      <OMI> 1 </OMI>
      <OMS name="litre_pre1964"
          cd="units_metric_obsolete1"/>
    </OMA>
    <OMA>
      <OMS name="times" cd="arith1"/>
      <OMA>
        <OMS name="divide" cd="arith1"/>
        <OMI> 1000028 </OMI>
        <OMI> 1000000 </OMI>
      </OMA>
      <OMS name="litre" cd="units_metric1"/>
    </OMA>
  </OMA>
</OMOBJ></FMP>

</CDDefinition>
</CD>
```

C.2.2 File: units_metric_obsolete1.sts

```
<CDSignatures
    xmlns="http://www.openmath.org/OpenMathCDS"
    type="sts" cd="units_metric_obselote1">
<CDSStatus> experimental </CDSStatus>

<Signature name="litre_pre1964" >
<OMOBJ xmlns="http://www.openmath.org/OpenMath">
  <OMS cd="dimensions1" name="volume"/>
</OMOBJ>
</Signature>

</CDSignatures>
```

C.2.3 File: units_imperial_obsolete1.ocd

```
<CD xmlns="http://www.openmath.org/OpenMathCD">
<CDName> units_imperial_obsolete1 </CDName>
<CDURL> http://www.openmath.org/cd/units_metric1.ocd
    </CDURL>
<CDReviewDate> 2008-03-31 </CDReviewDate>
<CDStatus> experimental </CDStatus>
<CDDate> 2008-03-31 </CDDate>
<CDVersion> 1 </CDVersion>
<CDRevision> 0 </CDRevision>

<Description>
This CD defines symbols to represent obsolete imperial
    standard measures.
</Description>

<CDDefinition>
<Name> rod </Name>
<Description>
This symbol represents the measure of one rod.
</Description>
<CMP> 1 rod = 5.5 yards </CMP>

<FMP><OMOBJ xmlns="http://www.openmath.org/OpenMath">
  <OMA>
    <OMS name="eq" cd="relation1"/>
    <OMA>
      <OMS name="times" cd="arith1"/>
      <OMI> 1 </OMI>
      <OMS name="rod" cd="units_imperial_obsolete1"/>
    </OMA>
    <OMA>
      <OMS name="times" cd="arith1"/>
      <OMA>
  <OMS name="divide" cd="arith1"/>
    <OMI> 11 </OMI>
  <OMI> 2 </OMI>
      </OMA>
      <OMS name="yard" cd="units_imperial1"/>
    </OMA>
  </OMA>
</OMOBJ></FMP>
</CDDefinition>

<CDDefinition>
<Name> pole </Name>
<Description>
This symbol represents the measure of one pole.
</Description>
<CMP> 1 pole = 1 rod </CMP>

<FMP><OMOBJ xmlns="http://www.openmath.org/OpenMath">
  <OMA>
    <OMS name="eq" cd="relation1"/>
    <OMA>
      <OMS name="times" cd="arith1"/>
      <OMI> 1 </OMI>
      <OMS name="pole" cd="units_imperial_obsolete1"/>
    </OMA>
    <OMA>
      <OMS name="times" cd="arith1"/>
      <OMI> 1 </OMI>
      <OMS name="rod" cd="units_imperial_obsolete1"/>
    </OMA>
  </OMA>
</OMOBJ></FMP>
</CDDefinition>

<CDDefinition>
<Name> perch </Name>
<Description>
This symbol represents the measure of one perch.
</Description>
<CMP> 1 perch = 1 rod </CMP>

<FMP><OMOBJ xmlns="http://www.openmath.org/OpenMath">
  <OMA>
    <OMS name="eq" cd="relation1"/>
    <OMA>
      <OMS name="times" cd="arith1"/>
      <OMI> 1 </OMI>
      <OMS name="perch" cd="units_imperial_obsolete1"/>
    </OMA>
    <OMA>
      <OMS name="times" cd="arith1"/>
      <OMI> 1 </OMI>
      <OMS name="rod" cd="units_imperial_obsolete1"/>
    </OMA>
  </OMA>
```

```
</OMOBJ></FMP>
</CDDefinition>

<CDDefinition>
<Name> chain </Name>
<Description>
This symbol represents the measure of one chain.
</Description>
<CMP> 1 chain = 4 rod </CMP>

<FMP><OMOBJ xmlns="http://www.openmath.org/OpenMath">
  <OMA>
    <OMS name="eq" cd="relation1"/>
    <OMA>
      <OMS name="times" cd="arith1"/>
      <OMI> 1 </OMI>
      <OMS name="chain" cd="units_imperial_obsolete1"/>
    </OMA>
    <OMA>
      <OMS name="times" cd="arith1"/>
      <OMI> 4 </OMI>
      <OMS name="rod" cd="units_imperial_obsolete1"/>
    </OMA>
  </OMA>
</OMOBJ></FMP>
</CDDefinition>

<CDDefinition>
<Name> league </Name>
<Description>
This symbol represents the measure of one league.
</Description>
<CMP> 1 league = 3 mile </CMP>

<FMP><OMOBJ xmlns="http://www.openmath.org/OpenMath">
  <OMA>
    <OMS name="eq" cd="relation1"/>
    <OMA>
      <OMS name="times" cd="arith1"/>
      <OMI> 1 </OMI>
      <OMS name="league" cd="units_imperial_obsolete1"/>
    </OMA>
    <OMA>
      <OMS name="times" cd="arith1"/>
      <OMI>3</OMI>
      <OMS name = "mile" cd="units_imperial1"/>
    </OMA>
  </OMA>
</OMOBJ></FMP>
</CDDefinition>

</CD>
```

C.2.4 File: units_imperial_obsolete1.sts

```
<CDSignatures
    xmlns="http://www.openmath.org/OpenMathCDS"
    type="sts" cd="units_imperial_obsolete1">
<CDSStatus> experimental </CDSStatus>
<Signature name="rod" >
<OMOBJ xmlns="http://www.openmath.org/OpenMath">
  <OMS cd="dimensions1" name="length"/>
</OMOBJ>
</Signature>
<Signature name="pole" >
<OMOBJ xmlns="http://www.openmath.org/OpenMath">
  <OMS cd="dimensions1" name="length"/>
</OMOBJ>
</Signature>
<Signature name="perch" >
<OMOBJ xmlns="http://www.openmath.org/OpenMath">
  <OMS cd="dimensions1" name="length"/>
</OMOBJ>
</Signature>
<Signature name="chain" >
<OMOBJ xmlns="http://www.openmath.org/OpenMath">
  <OMS cd="dimensions1" name="length"/>
</OMOBJ>
</Signature>
<Signature name="league" >
<OMOBJ xmlns="http://www.openmath.org/OpenMath">
  <OMS cd="dimensions1" name="length"/>
</OMOBJ>
</Signature>
</CDSignatures>
```

C.2.5 File: units_beer1.ocd

```
<CD xmlns="http://www.openmath.org/OpenMathCD">
<CDName> units_beer1 </CDName>
<CDURL> http://www.openmath.org/cd/units_metric1.ocd
    </CDURL>
<CDReviewDate> 2008-03-31 </CDReviewDate>
<CDStatus> experimental </CDStatus>
<CDDate> 2008-03-07 </CDDate>
<CDVersion> 1 </CDVersion>
<CDRevision> 1 </CDRevision>


<Description>
This CD defines symbols to represent the different sizes
    of beer cask. The current definition is used, unless
    otherwise stated.
</Description>

<CDDefinition>
<Name> firkin </Name>
<Description>
This symbol represents the measure of one firkin
</Description>

<FMP><OMOBJ xmlns="http://www.openmath.org/OpenMath">
<OMA>
  <OMS name="eq" cd="relation1"/>
  <OMS name="firkin" cd="units_beer1"/>
  <OMA>
    <OMS name="times" cd="arith1"/>
    <OMI>9</OMI>
    <OMS name="gallon" cd="units_imperial1"/>
  </OMA>
</OMA>
</OMOBJ></FMP>
</CDDefinition>

<CDDefinition>
<Name> kilderkin </Name>
<Description>
This symbol represents the measure of one kilderkin
</Description>

<FMP><OMOBJ xmlns="http://www.openmath.org/OpenMath">
<OMA>
  <OMS name="eq" cd="relation1"/>
  <OMS name="kilderkin" cd="units_beer1"/>
  <OMA>
    <OMS name="times" cd="arith1"/>
    <OMI>18</OMI>
    <OMS name="gallon" cd="units_imperial1"/>
  </OMA>
</OMA>
</OMOBJ></FMP>
</CDDefinition>

<CDDefinition>
<Name> barrel </Name>
<Description>
This symbol represents the measure of one barrel
</Description>

<FMP><OMOBJ xmlns="http://www.openmath.org/OpenMath">
<OMA>
  <OMS name="eq" cd="relation1"/>
  <OMS name="barrel" cd="units_beer1"/>
  <OMA>
    <OMS name="times" cd="arith1"/>
    <OMI>36</OMI>
    <OMS name="gallon" cd="units_imperial1"/>
  </OMA>
</OMA>
</OMOBJ></FMP>
</CDDefinition>

<CDDefinition>
<Name> hogshead </Name>
<Description>
This symbol represents the measure of one hogshead
</Description>

<FMP><OMOBJ xmlns="http://www.openmath.org/OpenMath">
<OMA>
```

```
80    <OMS name="eq" cd="relation1"/>
      <OMS name="hogshead" cd="units_beer1"/>
82    <OMA>
        <OMS name="times" cd="arith1"/>
84      <OMI>54</OMI>
        <OMS name="gallon" cd="units_imperial1"/>
86    </OMA>
    </OMA>
88  </OMOBJ></FMP>
    </CDDefinition>
90  </CD>
```

C.2.6 File: units_beer1.sts

```
    <CDSignatures
        xmlns="http://www.openmath.org/OpenMathCDS"
        type="sts" cd="units_metric2">
2   <CDSStatus> experimental </CDSStatus>
    <Signature name="firkin" >
4   <OMOBJ xmlns="http://www.openmath.org/OpenMath">
      <OMS cd="dimensions1" name="volume"/>
6   </OMOBJ>
    </Signature>
8   <Signature name="kilderkin" >
    <OMOBJ xmlns="http://www.openmath.org/OpenMath">
10    <OMS cd="dimensions1" name="volume"/>
    </OMOBJ>
12  </Signature>
    <Signature name="barrel" >
14  <OMOBJ xmlns="http://www.openmath.org/OpenMath">
      <OMS cd="dimensions1" name="volume"/>
16  </OMOBJ>
    </Signature>
18  <Signature name="hogshead" >
    <OMOBJ xmlns="http://www.openmath.org/OpenMath">
20    <OMS cd="dimensions1" name="volume"/>
    </OMOBJ>
22  </Signature>
    </CDSignatures>
```

Appendix D

Coding Standards

There are separate standards for the C# back-end part, and PHP front-end part, as well as XHTML and JavaScript for the front-end part. For all, good programming style must be maintained.

D.1 C# Coding Standards

Many of the specifics can be set up in the IDE to be adhered to automatically.

D.1.1 Bracketing Style

For consistency, all { and } will be on new lines on their own, and only the code within them will be indented, not the braces themselves. This is only not the case if the { and } are used for declaring an array, when they can be inline.

Case statements in general do not need {} round them. However, when a variable is declared within one, the whole case statement must have {} round it.

D.1.2 Tab Style

Anything after an opening { should be indented by one tab, until the corresponding }, which should be indented to the same level as its opener. Thus, case labels should be indented. However, in addition, the contents of a case statement should be indented.

D.1.3 Space Style

Spaces must be used on both sides of binary operators and after keywords in control flow statements to increase readability. They must not appear between a function name and its opening (. Spaces must follow semicolons in `for` conditions, commas in function parameters, and must precede and follow a : in a class declaration

D.1.4 Naming

Names must be meaningful and follow camelCase, with the follow exceptions/additions:

- Public methods, properties, and classes must have an initial capital, while private ones must not
- Parameters to methods must begin with px_, where x is:

  r read only variable

  w writable variable (passed with `ref` or `out` keyword)

- Class members names must begin with m_.

D.1.5 Conditionals

If a condition contains a constant, and an equality, this can appear on either side of the equality operator, because the C# compiler warns about potential accidental assignment.

D.2 PHP Coding Standards

D.2.1 Bracketing Style

For consistency, all { and } will be on new lines on their own, and only the code within them will be indented, not the braces themselves. This is only not the case if the { and } are used for declaring an array, when they can be inline.

Case statements in general do not need {} round them. However, when a variable is declared within one, the whole case statement must have {} round it.

D.2.2 Tab Style

Anything after an opening { should be indented by one tab, until the corresponding }, which should be indented to the same level as its opener. Thus, case labels should be indented. However, in addition, the contents of a case statement should be indented.

D.2.3 Space Style

Spaces must be used on both sides of binary operators and after keywords in control flow statements to increase readability. They must not appear between a function name and its opening (. Spaces must follow semicolons in `for` conditions, and commas in function parameters.

D.2.4 Naming

Names must be meaningful and follow camelCase, with the follow exceptions/additions:

- Parameters to methods must begin with \$px_, where x is:

  r read only variable

  w writable variable

- Arrays names must begin with "\$a_"

D.2.5 Conditionals

If a condition contains a constant, and an equality, this constant must appear on the left hand side of the equality operator, to avoid potential accidental assignment.

D.3 XHTML Coding Standards

The code must conform to XHTML Strict 1.0.

D.3.1 Tab Style

Anything after an opening <tag> should be indented by one tab, until the corresponding </tag>, which should be indented to the same level as its opener. <tags /> which self close therefore should not cause the next tag to be indented.

D.4 JavaScript Coding Standards

D.4.1 Bracketing Style

For consistency, all { and } will be on new lines on their own, and only the code within them will be indented, not the braces themselves. This is only not the case if the { and } are used for declaring an array, when they can be inline.

Case statements in general do not need {} round them. However, when a variable is declared within one, the whole case statement must have {} round it.

D.4.2 Tab Style

Anything after an opening { should be indented by one tab, until the corresponding }, which should be indented to the same level as its opener. Thus, case labels should be indented. However, in addition, the contents of a case statement should be indented.

D.4.3 Space Style

Spaces must be used on both sides of binary operators and after keywords in control flow statements to increase readability. They must not appear between a function name and its opening (. Spaces must follow semicolons in `for` conditions, and commas in function parameters.

D.4.4 Naming

Names must be meaningful and follow camelCase, with the follow exceptions/additions:

- Parameters to methods must begin with px_, where x is:

  r read only variable

  w writable variable

D.4.5 Conditionals

If a condition contains a constant, and an equality, this constant must appear on the left hand side of the equality operator, to avoid potential accidental assignment.

D.4.6 Semicolons

Semicolons must follow every statement, despite being optional in the language.

Appendix E

Test Plan

E.1 Introduction

Testing was split between back-end and front-end, followed by system testing.

E.2 Back-end Tests

In the end, we structured our test plan such that for each dimension that we had units for (and some that we did not), we tried to come up with tests to cover the following:

1. a metric unit to an imperial unit
2. a metric unit to a metric unit
3. an imperial unit to a metric unit
4. a metric unit to a US unit
5. a US unit to a metric unit
6. an imperial unit to a US unit
7. an imperial unit to an imperial unit
8. a US unit to an imperial unit
9. a US unit to a US unit
10. a prefixed unit to a non-prefixed unit
11. a non-prefixed unit to a prefixed unit
12. a prefixed unit to a prefixed unit
13. a named unit to a derived unit
14. a named unit to a named unit
15. a derived unit to a named unit
16. a derived unit to a derived unit
17. prefixed versions of named/derived combinations
18. to metric
19. to imperial
20. to U.S.

21. to new unit (without definition)
22. to new unit (with definition)
23. from new unit (without definition)
24. from new unit (with definition)
25. to & from new unit (without definitions)
26. to & from new unit (with definitions)
27. to cause all error codes to ensure appropriate data is also returned
28. to ensure both single-step and multi-step conversions work correctly

Using this scheme, we came up with over 200 tests for the back-end alone, which we felt covered a lot of the functionality. These tests are detailed in table E.1 on page 157. We could not find a test for every point for every dimension, however—for example, there are no special U.S. units for several of the dimensions, and therefore we could not write test cases involving these. Also, some tests covered several categories. Obviously, it is impossible to test for everything, but we felt that what we had did a reasonable job, influenced as it was by the knowledge of some of the system's limitations. To ease our testing exertions, we wrote a program to runs the tests. This program simply read a directory of text files containing test definitions[1], and executed each test, storing the result in another file, along with an indicator of Pass/Fail and the time taken. It was necessary to test that all error codes were returned correctly and resulted in the correct error message and optionally an offer of an action to take to resolve the problem, such as uploading files.

The requirements the back-end was supposed to fulfil are as follows: FR 1, 2, 5, 6, 7, 9; NFR 5, 6, 7, 8, 9 (parts), 10, 11, 12.

Now we will look at how these requirements are examined in these tests of the back-end.

FRs 1 (not 1c), 2 (but not part a), 5, and 6 are all tested in these tests. FR 1c cannot be tested until new units appear on the OpenMath website. FR 7 is not tested, because abbreviations are not used, although the pseudonym "tonne" is present for "megagramme", so small parts of it are tested. FR 9 is not tested in these tests, and a separate check to ensure the files are generated will have to be performed.

NFR 5 and NFR 6 are both exercised in these tests, timing data is stored, and the requirements explicitly exclude the parts where the user is choosing and uploading the file, since it has no control over these, and therefore an automated process to copy the files in will serve the purpose. NFR 7 is met by the code itself; it is hard to test that it uses the smallest

[1] in a simple format devised by ourselves, containing the input parameters, the expected output and the expected error code, but also with provision for commands to execute before and after the test—a typical use for these were to copy in any "user-generated" CDs required for the test, and delete them again afterwards, so each test was entirely self-contained

amount possible, because there is nothing to compare it to. However, as the design was specified to use the minimum amount of memory traded-off with ease of writing, it should pass this requirement. NFR 8 cannot be tested by these simple tests, which are designed to be run quickly. It will need to be verified separately. NFR 9 can only be tested in the back-end inasmuch as that sufficient information is output to make a meaningful error message. NFR 10 will be examined in these tests. Requirement NFR 11 is met by virtue of the back-end being written in C#, therefore is not tested. NFR 12 can only be verified by checking the code itself.

As these tests are for the back-end of the system, underscores are required between units in a compound unit name.

Table E.1: Tests and Expected Outputs

#	Source	Destination	Additional CDs	ErrorCode	Output
			Length		
1	11.5 metre	foot		0	37.7296587926509
2	11.5 mile	picometre		0	1.8507456E+16
3	11.5 kilometre	mile_us_survey		0	7.14575441919192
4	11.5 mile	yard_us_survey		0	20239.95952
5	11.5 mile_us_survey	foot		0	60720.1214402429
6	11.5 mile_us_survey	yard_us_survey		0	20240
7	11.5 metre	petametre		0	1.15E-14
8	11.5 picometre	nanometre		0	0.0115
9	11.5 yard_us_survey	kilometre		0	0.0105156210312421
10	11.5 metre	rod		1	rod
11	11.5 metre	rod	units_imperial_obsolete1.ocd	0	2.28664598743339
12	11.5 chain	metre		1	chain
13	11.5 chain	metre	units_imperial_obsolete1.ocd	0	231.3432
14	11.5 chain	rod		1	chain
15	11.5 chain	rod	units_imperial_obsolete1.ocd	0	46
16	17.3 yard	imperial		1	imperial
17	5 mile	metric		0	8.04672 kilometre
18	8 kilometre	imperial		0	4 mile 7 furlong 168 yard 2 foot 8 inch 629.921259821388 mil
19	8 kilometre	imperial	units_imperial_obsolete1.ocd	0	1 league 1 mile 7 furlong 7 chain 2 pole 3 yard 2 foot 8 inch 629.92125982231 mil
20	8 kilometre	US		0	4 mile_us_survey 1708 yard_us_survey 2.66666666666817 foot_us_survey
			Mass		
21	17 gramme	pound_mass		0	0.0374785845714292
22	17 ounce	kilogramme		0	0.481941893125
23	10 millitonne	pound_mass		1	millitonne
24	10 millitonne	pound_mass	units_metric2.ocd	0	22.0462262184878
25	10 millitonne	kilogramme	units_metric2.ocd	0	10
26	10 millitonne	stone	units_metric2.ocd	0	1.5747304441777
27	10.5 pound_mass	stone		0	0.75
28	10 ton_long	metric		0	10.160469088 megagramme
29	10 ton_long	metric	units_metric2.ocd	0	1.0160469088 dekatonne
30	10 kilogramme	imperial		0	1 stone 8 pound_mass 0.739619495804108 ounce
31	10.5 kilogramme	US		7	
32	17 milligramme	millipound_mass		1	millipound_mass
33	17 milligramme	pound_mass		0	3.74785845714292E-05
34	10.5 pound_mass	US		7	

#	Source	Destination	Additional CDs	ErrorCode	Output
35	10 millitonne	megatonne		1	millitonne
36	10 millitonne	megatonne	units_metric2.ocd	0	1E-08
37	10 pound_mass	megatonne		1	megatonne
Time					
38	15 second	hour		0	0.00416666666666667
39	15 day	hour		0	360
40	15 terasecond	week		0	24801587.3015873
41	15 megasecond	gigasecond		0	0.015
42	15 day	gigasecond		0	0.001296
43	15 second	millisecond		0	15000
44	10 calendar_year	calendar_month		0	120
45	10 calendar_month	calendar_year		0	0.833333333333333
46	10 day	calendar_month		5	
47	10 calendar_month	day		5	
48	10 calendar_year	day		5	
49	1000 day	calendar_year		5	
50	15 day	metric		0	1.296 megasecond
51	15 megasecond	time		0	24 week 5 day 14 hour 39 minute 59.9999999999021 second
52	15 hour	metric		0	0.054 megasecond
53	15 hour	metric		0	0.054 megasecond
54	15 hour	US		7	
Temperature					
55	75 degree_Celsius	degree_Fahrenheit		0	167
56	75 degree_Fahrenheit	degree_Celsius		0	23.8888888888889
57	-40 degree_Fahrenheit	degree_Celsius		0	-40
58	212 degree_Celsius	degree_Kelvin		0	485.15
59	212 degree_Fahrenheit	degree_Kelvin		0	373.15
60	10 megadegree_Kelvin	millidegree_Kelvin		0	10000000000
61	10 degree_Celsius	millidegree_Kelvin		0	283150
62	10 millidegree_Kelvin	degree_Celsius		0	-273.14
63	10 degree_Fahrenheit	metric		0	2.60927777777778 hectodegree_kelvin
64	10 degree_Celsius	imperial		0	50 degree_Fahrenheit
65	10 degree_Celsius	US		7	
Current					
66	10 Amp	milliamp		0	10000
67	10 megaamp	Amp		0	10000000
68	10 megaamp	microamp		0	10000000000000
69	10 megaamp	metric		1	metric
70	10 megaamp	Imperial		7	
71	10 megaamp	US		7	
Area					

#	Source	Destination	Additional CDs	ErrorCode	Output
72	10 acre	mile_sqrd		0	0.015625
73	6 mile_sqrd	acre		0	3840
74	10 metre_sqrd	acre		0	0.00247105381467165
75	10 metre_sqrd	centimetre_sqrd		0	100000
76	10 centimetre_sqrd	metre_sqrd		0	0.001
77	10 centimetre_sqrd	kilometre_sqrd		0	1E-09
78	10 acre_us_survey	acre		0	10.00004000012
79	10 acre_us_survey	kilometre_sqrd		0	0.0404687260987425
80	10 kilometre_sqrd	imperial		0	2471.05381467165 acre
81	10 acre	metric		7	
82	10 kilometre_sqrd	US		0	2471.04393046628 acre_us_survey
83	10 acre	petametre_sqrd		0	4.0468564224E-26
84	10.2 gigametre_sqrd	acre		0	2.52047489096509E+15
85	10.8 kilometre_sqrd	acre_us_survey		0	2668.72744490358
86	1.02 acre	yard_us_survey_sqrd		0	4936.78025281975
87	19.2 yard_us_survey_sqrd	acre_us_survey		0	0.00396694214876033
			Volume		
88	6 litre	pint		0	10.5585239183562
89	6 pint	litre		0	3.4095675
90	6 pint	gallon		0	0.75
91	105 barrel	pint		1	barrel
92	106 barrel	pint	units_beer1.ocd	0	30528
93	106 barrel	kilderkin	units_beer1.ocd	0	212
94	10 kilderkin	kilolitre	units_beer1.ocd	0	0.8182962
95	100 litre	acre_foot		0	8.10713193789912E-05
96	100 acre_foot	litre		0	123348183.754752
97	10 litre	decimetre_cubed		0	10
98	10 decimetre_cubed	litre		0	10
99	6 litre	litre_pre1964		1	litre_pre1964
100	6 litre	litre_pre1964	units_metric_obsolete1.ocd	0	5.99983200470387
101	1.25 litre	yard_cubed		0	0.00163493827414299
102	1.25 litre	foot_cubed		0	0.0441433334018607
103	1.25 foot_cubed	litre		0	35.39605824
104	1.25 foot_cubed	yard_cubed		0	0.0462962962962963
105	1.25 mile_cubed	pint		0	9168718229160.83
106	1.25 mile_us_survey_cubed	pint		0	9168773241690.26
107	1.25 metre_cubed	yard_us_survey_cubed		0	1.63492846453296
108	1.25 mile_us_survey_cubed	metre_cubed		0	5210258543.28946
109	1.25 mile_cubed	mile_us_survey_cubed		0	1.249992500015
110	1.25 mile_us_survey_cubed	yard_us_survey_cubed		0	6814720000
111	6 pint	metric		0	0.34095675 dekalitre
112	17.3 litre	metric		1	metric

#	Source	Destination	Additional CDs	ErrorCode	Output
113	17.3 litre	imperial		0	3 gallon 6 pint 8.8748792918749 3 fluid_ounce
114	17.3 litre	US		0	3 gallon_us_dry 1 gallon_us_liquid 1 cup_us 2.14207326136443 fluid_ounce_us
115	10 decimetre_cubed	hectolitre		0	0.1
116	105 barrel	kilderkin		1	barrel
			Speed		
117	6 metre_per_second	mile_per_hour		0	13.4216177523264
118	6 metre_per_second	centimetre_per_minute		0	36000
119	30 mile_per_hour	centimetre_per_minute		0	80467.2
120	6 kilometre_per_second	mile_per_hour		0	13421.6177523264
121	6 metre_per_second	mile_us_survey_per_hour		0	13.4215909090909
122	6 foot_per_second	mile_per_day		0	98.1818181818182
123	6 millimetre_per_second	centimetre_per_minute		0	36
124	30 inch_per_millisecond	mile_us_survey_per_hour		0	1704.54204545455
125	30 mile_us_survey_per_hour	inch_per_millisecond		0	0.528001056002112
126	30 mile_us_survey_per_hour	centimetre_per_minute		0	80467.3609347219
127	30 mile_us_survey_per_hour	yard_us_survey_per_millisecond		0	0.0146666666666667
128	30 inch_per_millisecond	metric		7	
129	30 metre_per_second	imperial		7	
130	30 inch_per_millisecond	US		7	
131	30 inch_per_millisecond	mile_per_hour		0	1704.54545454545
			Acceleration		
132	17.3 metre_per_second_sqrd	mile_per_hour_sqrd		0	139316.392269148
133	17 foot_per_second_sqrd	mile_per_hour_sqrd		0	41727.2727272727
134	17.3 milli-metre_per_nanosecond_sqrd	mile_per_hour_sqrd		0	1.39316392269148E+20
135	17.3 milli-metre_per_nanosecond_sqrd	kilometre_per_hour_sqrd		0	2.24208E+20
136	17.3 yard_per_day_sqrd	foot_us_survey_per_week_sqrd		0	2543.0949138
137	17.3 foot_us_survey_per_week_sqrd	yard_per_day_sqrd		0	0.117687310204552
138	17.3 foot_us_survey_per_week_sqrd	yard_us_survey_per_day_sqrd		0	0.117687074829932
139	17.3 me-tre_per_nanosecond_sqrd	kilometre_per_hour_sqrd		0	2.24208E+23
140	17.3 mile_per_hour_sqrd	metric		0	2.14827555555556 milli-metre_per_second_sqrd
141	17.3 metre_per_second_sqrd	imperial		0	1.07497216257059E-02 mile_per_second_sqrd
142	17.3 rod_per_minute_sqrd	chain_per_second_sqrd		1	rod_per_minute_sqrd

#	Source	Destination	Additional CDs	ErrorCode	Output
143	17.3 rod_per_minute_sqrd	chain_per_second_sqrd	units_imperial_obsolete1.ocd	0	0.00120138888888889
144	17.3 metre_per_second_sqrd	US		0	17.300034600069 metre_us_survey_per_second_squared
145	17.3 mile_us_survey_per_hour_sqrd	metre_per_second_sqrd		0	0.00214827985211526
			Force		
146	6 Newton	pound_force		0	1.34892086330935
147	6 pound_force	Newton		0	26.688
148	6 Newton	meganewton		0	6E-06
149	6 meganewton	Newton		0	6000000
150	6 meganewton	giganewton		0	0.006
151	6 meganewton	kilogramme_metre_per_second_sqrd		0	600000
152	6 kilogramme_metre_per_second_sqrd	meganewton		0	6000000
153	6 meganewton	imperial		0	1348920.86330935 pound_force
154	6 pound_force	metric		0	2.6688 dekanewton
155	6 pound_force	US		7	
156	6 Newton	US		7	
157	6 ounce_inch_per_minute_sqrd	kilogramme_metre_per_second_sqrd		0	1.20012981229167E-06
			Pressure		
158	105.36745 Newton_per_centimetre_sqrd	Pascal		0	1053674.5
159	17.3 kilonewton_per_rod_sqrd	Pascal	units_imperial_obsolete1.ocd	0	683.987695901114
160	17.3 kilonewton_per_rod_sqrd	kilopascal	units_imperial_obsolete1.ocd	0	0.683987695901114
161	17.3 pound_force_per_rod_sqrd	kilopascal	units_imperial_obsolete1.ocd	0	3.04252885347956E-03
162	17.3 pound_force_per_mile_sqrd	metric		0	29.7107155407046 micropascal
163	17.3 bar	metric		0	1.73 megapascal
164	17.3 Pascal	imperial		0	173 microbar
165	17.3 bar	US		7	
166	105 Newton_per_inch_sqrd	Pascal		0	162750.325500651
167	17.3 pound_force_per_mile_sqrd	millibar		0	2.971218504552E-07
168	105.36745 Pascal	Newton_per_centimetre_sqrd		0	0.010536745
169	105.36745 Pascal	kilopascal		0	0.10536745
170	105.36745 Newton_per_centimetre_sqrd	Newton_per_metre_sqrd		0	1053674.5
			Energy		
171	10 Joule	microjoule		0	10000000
172	10 megajoule	Joule		0	10000000
173	10 megajoule	microjoule		0	10000000000000

#	Source	Destination	Additional CDs	ErrorCode	Output
174	10 megajoule	Watt_second		0	10000000
175	10 megajoule	megawatt_second		0	10
176	10 megajoule	metric		1	metric
177	10 megajoule	imperial		7	
178	10 megajoule	US		7	
			Power		
179	19 Joule_per_second	Watt		0	19
180	19 megajoule_per_second	watt		0	19000000
181	19 Joule_per_second	kilowatt		0	0.019
182	19 kilowatt	megajoule_per_second		0	0.019
183	19 kilowatt	joule_per_second		0	19000
184	19 Joule_per_second	metric		1	metric
185	19 Joule_per_second	Imperial		7	
186	19 Joule_per_second	US		7	
187	19.1 Joule_per_second	megajoule_per_hour		0	0.06876
			Voltage		
188	10 Volt	millivolt		0	10000
189	10 megavolt	Volt		0	10000000
190	10 megavolt	microvolt		0	10000000000000
191	10 megavolt	metric		1	metric
192	10 megavolt	Imperial		7	
193	10 megavolt	US		7	
			Charge		
194	10 Coulomb	millicoulomb		0	10000
195	10 megacoulomb	Coulomb		0	10000000
196	10 megacoulomb	microcoulomb		0	10000000000000
197	10 megacoulomb	metric		1	metric
198	10 megacoulomb	Imperial		7	
199	10 megacoulomb	US		7	
			Miscellaneous		
200	17 gramme	pound_force		4	
201	105 Newton_per_inch_cubed	Pascal		4	
202	105 Newton_per_centimetre_cubed	pound_force_per_foot_cubed		0	668450.740143885
203	17.3 rod_per_minute_sqrd	chain_per_second		1	rod_per_minute_sqrd
204	17.3 rod_per_minute_sqrd	chain_per_second	units_imperial_obsolete1.ocd	4	
205	10 calendar_year	day	units_imperial1.ocd	10	units_imperial1
206	10 calendar_year	day	units_imperial1.ocd	10	units_imperial1
207	10 calendar_year	day	units_imperial_broken1.ocd	11	XML error in units_imperial_broken1.ocd:Unexpected end of file while parsing Comment has occurred. Line 46, position 1.

#	Source	Destination	Additional CDs	ErrorCode	Output
208	10a calendar_year	day		2	`Your command line: --source_quantity 10a --source_unit calendar_year --destination_unit day Usage: OpenMathConverter.exe --source_quantity <amount> --source_unit <name> --destination_unit <name> Where <amount> is a number.`
209	212 relative_degree_Fahrenheit	relative_degree_Kelvin		0	117.777777777778
210	100 relative_degree_Celsius	relative_degree_Fahrenheit		0	180

E.2.1 Test Justification

For many of the tests in table E.1, it is obvious why they are being performed. However, some merit additional explanation, which will be covered in this section.

Tests 18 and 19 are to ensure that when the system is given a new CD and then asked to perform a conversion to a measurement standard, the new units are used.

Tests 46–49 are expected to not successfully convert, because calendar_year contains two part definitions, one in terms of months (exactly 12), and one in terms of days (interval 365 366). Both of these are necessary, but it is not clear how best to handle two FMPs for a single unit, nor is it clear how to handle intervals (see Appendix H). This also means that calendar_month, whose definition is an interval of days, also cannot be processed correctly.

E.3 Front-end Tests

Once the back-end tests are complete, the front-end needs to be tested. The reason that this must come after the back-end has been shown to work (or at least any problems revealed) is that much of the front-end functionality relies on the back-end—for example, the significant figures control can only affect what the back-end returns. Therefore, the front-end tests tend to be more informal; there is not a clear distinction in most cases between testing the front-end only, and testing the whole system. Therefore our test plan tends to run integration tests on the whole system, by using the front-end in its intended capacity as an interface to the back-end.

The requirements that need to be tested specifically for the front-end are: FR 1, 2, 3, 5, 7, 9, 10, NFR 5, 6, 7, 9 and 12, and DR 1. However, when we do integration testing, all the tests that relate to either the front- or back-end will need to be tested.

Similarly to the back-end, several features not specified in the requirements had been added to the front-end, and therefore these were tested in addition to the requirements. Examples of this include the “Reverse this conversion” option.

To test the whole system, we elected to attempt several standard conversions, and then attempt to cause all the different error messages to appear, to ensure that not only did the message appear but that appropriate actions could be taken to resolve the issue. The results of all the testing carried out are included in section F.2.

E.4 System Tests

These were mostly covered by the Front-end tests, and such are often grouped together in the results.

Appendix F

Test Results

F.1 Back-End Test Results

F.1.1 Main Tests Results

The raw data used to generate these tables can be found in ProjectTester directory on the CD included with the submission—which additionally includes the program used to run the tests. The data in these tables was generated directly by the program. It may be noted that test 208 failed in the raw data. This is because the testing program was not sufficiently flexible to allow an invalid command line, so the test failed. It had to be run manually independently of the test program, and was found to pass.

Table F.1: Test Results

#	ErrorCode	Output	Pass/Fail?	Time (s)
Length				
1	0	37.7296587926509	Passed	2.693
2	0	1.8507456E+16	Passed	3.444
3	0	7.14575441919192	Passed	2.814
4	0	20239.95952	Passed	3.94
5	0	60720.1214402429	Passed	2.763
6	0	20240	Passed	2.693
7	0	1.15E-14	Passed	2.683
8	0	0.0115	Passed	2.693
9	0	0.0105156210312421	Passed	3.124
10	1	rod	Passed	2.894
11	0	2.28664598743339	Passed	2.703
12	1	chain	Passed	3.204
13	0	231.3432	Passed	2.693
14	1	chain	Passed	3.34
15	0	46	Passed	2.904
16	1	imperial	Passed	3.174
17	0	8.04672 kilometre	Passed	2.794
18	0	4 mile 7 furlong 168 yard 2 foot 8 inch 629.921259821388 mil	Passed	3.4
19	0	1 league 1 mile 7 furlong 7 chain 2 pole 3 yard 2 foot 8 inch 629.92125982231 mil	Passed	3.164
20	0	4 mile_us_survey 1708 yard_us_survey 2.66666666666817 foot_us_survey	Passed	3.24
Mass				
21	0	0.0374785845714292	Passed	2.804
22	0	0.481941893125	Passed	2.884
23	1	millitonne	Passed	2.713
24	0	22.0462262184878	Passed	2.934
25	0	10	Passed	3.134
26	0	1.5747304441777	Passed	2.814
27	0	0.75	Passed	2.673
28	0	10.160469088 megagramme	Passed	2.804
29	0	1.0160469088 dekatonne	Passed	3.755
30	0	1 stone 8 pound_mass 0.739619495804108 ounce	Passed	3.24
31	7		Passed	2.924
32	1	millipound_mass	Passed	2.904
33	0	3.74785845714292E-05	Passed	2.673
34	7		Passed	2.784
35	1	millitonne	Passed	2.673
36	0	1E-08	Passed	2.713
37	1	megatonne	Passed	2.854

#	ErrorCode	Output	Pass/Fail?	Time (s)
		Time		
38	0	0.00416666666666667	Passed	2.693
39	0	360	Passed	2.693
40	0	24801587.3015873	Passed	2.964
41	0	0.015	Passed	2.884
42	0	0.001296	Passed	2.693
43	0	15000	Passed	3.4
44	0	120	Passed	2.904
45	0	0.833333333333333	Passed	2.834
46	5		Passed	2.703
47	5		Passed	2.693
48	5		Passed	2.693
49	5		Passed	2.703
50	0	1.296 megasecond	Passed	2.773
51	0	24 week 5 day 14 hour 39 minute 59.9999999999021 second	Passed	3.184
52	0	0.054 megasecond	Passed	2.784
53	0	0.054 megasecond	Passed	3.244
54	7		Passed	2.934
		Temperature		
55	0	167	Passed	2.703
56	0	23.8888888888889	Passed	2.703
57	0	-40	Passed	2.884
58	0	485.15	Passed	2.703
59	0	373.15	Passed	2.693
60	0	10000000000	Passed	2.683
61	0	283150	Passed	2.773
62	0	-273.14	Passed	2.713
63	0	2.60927777777778 hectodegree_kelvin	Passed	3.765
64	0	50 degree_Fahrenheit	Passed	2.713
65	7		Passed	2.683
		Current		
66	0	10000	Passed	2.924
67	0	10000000	Passed	2.683
68	0	10000000000000	Passed	2.673
69	1	metric	Passed	2.844
70	7		Passed	2.844
71	7		Passed	3.44
		Area		
72	0	0.015625	Passed	3.44
73	0	3840	Passed	3.34
74	0	0.00247105381467165	Passed	2.693
75	0	100000	Passed	2.703
76	0	0.001	Passed	2.703
77	0	1E-09	Passed	2.804
78	0	10.00004000012	Passed	2.673
79	0	0.0404687260987425	Passed	2.703
80	0	2471.05381467165 acre	Passed	2.814
81	7		Passed	2.673
82	0	2471.04393046628 acre_us_survey	Passed	2.703
83	0	4.0468564224E-26	Passed	2.713
84	0	2.52047489096509E+15	Passed	2.713
85	0	2668.72744490358	Passed	2.834
86	0	4936.78025281975	Passed	2.713
87	0	0.00396694214876033	Passed	2.703
		Volume		
88	0	10.5585239183562	Passed	2.733
89	0	3.4095675	Passed	2.964
90	0	0.75	Passed	3.124

#	ErrorCode	Output	Pass/Fail?	Time (s)
91	1	barrel	Passed	2.663
92	0	30528	Passed	2.814
93	0	212	Passed	2.733
94	0	0.8182962	Passed	2.683
95	0	8.10713193789912E-05	Passed	3.24
96	0	123348183.754752	Passed	2.914
97	0	10	Passed	2.854
98	0	9.99999999999999	FAILED	2.723
99	1	litre_pre1964	Passed	2.794
100	0	5.99983200470387	Passed	2.964
101	0	0.00163493827414299	Passed	3.204
102	0	0.0441433334018607	Passed	2.894
103	0	35.39605824	Passed	3.144
104	0	0.0462962962962963	Passed	3.24
105	0	9168718229160.83	Passed	2.854
106	0	9168773241690.26	Passed	3.4
107	0	1.63492846453296	Passed	3.154
108	0	5210258543.28946	Passed	2.844
109	0	1.249992500015	Passed	3.134
110	0	6814720000	Passed	2.743
111	0	0.34095675 dekalitre	Passed	3.224
112	1	metric	Passed	2.663
113	0	3 gallon 6 pint 8.87487929187493 fluid_ounce	Passed	2.804
114	0	3 gallon_us_dry 1 gallon_us_liquid 1 cup_us 2.14207326136443 fluid_ounce_us	Passed	3.274
115	0	0.1	Passed	2.854
116	1	barrel	Passed	2.673
Speed				
117	0	13.4216177523264	Passed	2.794
118	0	36000	Passed	2.864
119	0	80467.2	Passed	3.124
120	0	13421.6177523264	Passed	2.693
121	0	13.4215909090909	Passed	2.703
122	0	98.1818181818182	Passed	2.723
123	0	36	Passed	2.773
124	0	1704.54204545455	Passed	2.994
125	0	0.528001056002112	Passed	3.144
126	0	80467.3609347219	Passed	3.44
127	0	0.0146666666666667	Passed	2.693
128	7		Passed	3.64
129	7		Passed	2.874
130	7		Passed	2.904
131	0	1704.54545454545	Passed	2.693
Acceleration				
132	0	139316.392269148	Passed	2.894
133	0	41727.2727272727	Passed	2.723
134	0	1.39316392269148E+20	Passed	3.4
135	0	2.24208E+20	Passed	2.854
136	0	2543.0949138	Passed	2.733
137	0	0.117687310204552	Passed	2.914
138	0	0.117687074829932	Passed	2.944
139	0	2.24208E+23	Passed	2.703
140	7		FAILED	2.844
141	7		FAILED	2.984
142	1	rod_per_minute_sqrd	Passed	3.144
143	0	0.00120138888888889	Passed	2.713
144	7		FAILED	2.864
145	0	0.00214827985211526	Passed	2.703
Force				

#	ErrorCode	Output	Pass/Fail?	Time (s)
146	0	1.34892086330935	Passed	2.673
147	0	26.688	Passed	2.844
148	0	6E-06	Passed	3.104
149	0	6000000	Passed	2.864
150	0	0.006	Passed	3.124
151	4		FAILED	2.784
152	4		FAILED	3.4
153	0	1348920.86330935 pound_force	Passed	2.683
154	0	2.6688 dekanewton	Passed	2.743
155	7		Passed	2.663
156	7		Passed	2.683
157	0	1.20012981229167E-06	Passed	2.733
Pressure				
158	0	1053674.5	Passed	2.964
159	0	683.987695901114	Passed	2.834
160	0	0.683987695901114	Passed	2.924
161	0	0.00304237727136815	Passed	2.763
162	0	29.7107155407046 micropascal	Passed	4.75
163	0	1.73 megapascal	Passed	2.984
164	0	173 microbar	Passed	3.84
165	7		Passed	2.864
166	0	162750.325500651	Passed	2.723
167	5		FAILED	2.964
168	0	0.010536745	Passed	3.14
169	0	0.10536745	Passed	2.703
170	0	1053674.5	Passed	2.723
Energy				
171	0	10000000	Passed	2.693
172	0	10000000	Passed	2.974
173	0	10000000000000	Passed	2.663
174	4		FAILED	2.693
175	4		FAILED	2.784
176	1	metric	Passed	2.884
177	7		Passed	2.944
178	7		Passed	3.84
Power				
179	0	19	Passed	3.24
180	0	19000000	Passed	3.54
181	0	0.019	Passed	3.34
182	5		FAILED	2.964
183	5		FAILED	3.114
184	1	metric	Passed	2.663
185	7		Passed	2.683
186	7		Passed	2.794
187	0	0.06876	Passed	2.713
Voltage				
188	0	10000	Passed	2.964
189	0	10000000	Passed	3.14
190	0	10000000000000	Passed	2.974
191	1	metric	Passed	2.663
192	7		Passed	2.683
193	7		Passed	2.904
Charge				
194	0	10000	Passed	2.884
195	0	10000000	Passed	3.104
196	0	10000000000000	Passed	2.673
197	1	metric	Passed	2.683
198	7		Passed	2.663
199	7		Passed	2.994

#	ErrorCode	Output	Pass/Fail?	Time (s)
Miscellaneous				
200	4		Passed	3.64
201	4		Passed	2.914
202	0	668450.740143885	Passed	2.984
203	1	rod_per_minute_sqrd	Passed	3.44
204	4		Passed	2.713
205	10	units_imperial1	Passed	2.453
206	10	units_imperial1	Passed	2.463
207	11	XML error in units_imperial_broken1.ocd:Unexpected end of file while parsing Comment has occurred. Line 46, position 1.	Passed	2.453
208	2	`Your command line: --source_quantity 10a --source_unit calendar_year --destination_unit day Usage: OpenMathConverter.exe --source_quantity <amount> --source_unit <name> --destination_unit <name> Where <amount> is a number.`	Passed	3.84
209	0	117.777777777778	Passed	2.683
210	0	180	Passed	2.733

Further to these, a test was undertaken whereby the local copy of the data files were deleted and then the tests were run again. The first test case in this run took significantly longer, but still took under 10 seconds. Following this, all other test cases took a comparable time to before.

As requirement FR 9 had not been shown to pass from these tests alone, because the back-end part generates the text file, it was also checked that these files were indeed generated, but deleting them from the system and running the program again. It was found that the system writes the names to the three files, units_prefixable.txt, units_not_prefixable.txt and prefixes.txt, after any conversion attempt, except when the unit definition files could not be loaded.

F.1.2 Summary of Requirements comparison for Back-End

In this section, we will review the various requirements, and whether we feel the back-end passed those that we feel it should. Will examine these results in the main text.

Table F.2: Summary of whether back-end met its requirements

Requirement	Result	Notes
Functional Requirements		
1	Pass	The system works correctly in many cases, but fails for some specific types
1(a)	Pass	
1(a)i	Pass	When a unit is not recognised, a specific error code, along with the unit name, is returned
1(a)i.A	N/A	This is covered by the front-end
1(b)	Pass	The system works out for itself which files to load
1(c)	N/A	This requirement cannot be tested, as there is no write access to the OpenMath website. However, it is believed that if the symbols get added to cdindex.html, they will be found.

2	Pass	New CDs were used successfully
2(a)	Pass	If the CD shares a name with an uploaded one, the conversion cannot occur until the contributed one is removed
2(a)i	N/A	The front-end will deal with uploads at all times
2(a)ii	N/A	This was ruled out during the design phase
2(b)	Pass	Definitions which share a name in different CDs are allowed, for example there are two different `seconds`
2(b)i	Fail	The system decides for itself, as it was felt the user should not be troubled with the detail.
2(b)i.A	N/A	
2(b)ii	-	The system allows both definitions to co-exist
2(c)	Pass	
2(c)i	Pass	
5	Pass	An error code is reserved for this
5(a)	Pass	Error-dependent information is returned
5(a)i	N/A	This is left for the front-end
6	Pass	
6(a)	Pass	The system uses the values with the smallest absolute logarithm
6(a)i	Pass	
9	Pass	The back-end generates the files used by the front-end for this purpose
10	N/A	This is performed by the front-end
Non-Functional Requirements		
5	Pass	Results are returned typically within 3 seconds
5(a)	Pass	Depending on the problem, sometimes the return is very rapid, but it never is more than 5 seconds
6	Pass	Time taken is in fact comparable to built-in units
6(a)	Pass	
6(b)	Pass	
7	Pass	
7(a)	N/A	
7(b)	Pass	
8	Pass	Uploaded files are deleted
8(a)	Pass	The files are deleted when they are more than 30 minutes old
9	Pass	Although not really covered by the back-end, it supplies enough information to make useful messages
10	Pass	Changes to the set of known units causes the result to change accordingly
11	Pass	C# is used
12	N/A	This has not been tested, but standards have been followed

F.2 Front-End Test Results

Firstly, tests were carried out to ensure that the full range of error messages could be displayed, and that messages were informative. As this is for the front-end of the system, spaces can be used to separate units in a compound unit name.

Table F.3: Front-End Error Testing

Input	**Error Intended**	**Message Returned**
30 mile per hour to metre per second	No Error	30 mile per hour is 13.411 metre per second.
2 metres to foot	Unknown unit	A unit you entered—metres—was unknown. Please provide the file which contains the definition of this unit.
1# metre to foot	Invalid command line	Fatal error executing the command:`Your command line: --source_quantity 1# --source_unit metre --destination_unit foot Usage: OpenMathConverter.exe --source_quantity <amount> --source_unit <name> --destination_unit <name> Where <amount> is a number.`
1 metre to gallon	Different dimensions	The two units were known, but found to be different dimensions. Please supply a definition for the units to reconcile this difference, remembering to ensure the new definition has a different name.
units_imperial_broken1.ocd	Invalid XML, file deleted	XML error in units_imperial_broken1.ocd:Unexpected end of file while parsing Comment has occurred. Line 46, position 1. units_imperial_broken1.ocd deleted. Please try conversion again
No local files, no network connection	Graphs not loadable	Could not load the units into the program because of an error.
1 day to calendar_year	Conversion not found	A route to convert between the two units could not be found. This means that this program is missing a definition that would allow it to convert between the two units.
1 day to U.S.	Unit Type not found	There were no units of that standard in that dimension.
units_imperial1	Duplicate CD	A duplicate CD name was found: units_imperial1 units_imperial1.ocd deleted. Please try another file. units_imperial1.sts deleted. Please try another file.
units_beer1	Upload confirmation	units_beer1.ocd uploaded successfully. units_beer1.sts uploaded successfully.
units_beer1 (again)	Error relating to overwriting	Uploading a units_beer1.ocd would overwrite an existing file. Please choose a different file or rename.Uploading an units_beer1.sts would overwrite an existing file. Please choose a different file or rename.

As stated in Section 5.2 of Appendix H, it has not yet been decided whether the significant figures control should affect "to measurement standard" conversions, and as such it does not. However, these tables show the effect of the control on a typical conversion, compared to the output from a "to imperial" conversion.

Conversion	
6 pint to litre	
Sig. Fig.	**Output**
15	3.4095675
14	3.4095675
13	3.4095675
12	3.4095675
11	3.4095675
10	3.4095675
9	3.4095675
8	3.4095675
7	3.409568
6	3.40957
5	3.4096
4	3.41
3	3.41
2	3.4
1	3

Conversion	
2 metre to imperial	
Sig. Fig.	**Output**
15	2 metre is 2 yard 6 inch 740.157480314952 mil
14	2 metre is 2 yard 6 inch 740.157480314952 mil
13	2 metre is 2 yard 6 inch 740.157480314952 mil
12	2 metre is 2 yard 6 inch 740.157480314952 mil
11	2 metre is 2 yard 6 inch 740.157480314952 mil
10	2 metre is 2 yard 6 inch 740.157480314952 mil
9	2 metre is 2 yard 6 inch 740.157480314952 mil
8	2 metre is 2 yard 6 inch 740.157480314952 mil
7	2 metre is 2 yard 6 inch 740.157480314952 mil
6	2 metre is 2 yard 6 inch 740.157480314952 mil
5	2 metre is 2 yard 6 inch 740.157480314952 mil
4	2 metre is 2 yard 6 inch 740.157480314952 mil
3	2 metre is 2 yard 6 inch 740.157480314952 mil
2	2 metre is 2 yard 6 inch 740.157480314952 mil
1	2 metre is 2 yard 6 inch 740.157480314952 mil

Notice how, in the specific conversion, the outputs for 3 and 4 significant figures are identical, because trailing zeros are dropped, as might be expected. In the "to standard" conversion, the result is the same for all.

F.2.1 Problem with Front-end

Having tried a wide range of conversions, it has been found that the front-end largely works as designed. One difference, which is in fact a bug, is that if the upload units button is clicked in order to manually upload units, it then must be clicked again to remove the boxes from the display, whether or not a file is selected—attempts to allow the user to use the convert button as well failed. However, when the mode is automatically triggered the Upload and Convert button works as expected.

F.2.2 Illustrative Conversions

To demonstrate one the powers of the system, some quite extreme conversions will now be demonstrated.

Input	**Output**
1000000000000000000000000000000000000 metre in yard	1.0936132983377E+36 yard.
1e-36 metre in yard	1.0936132983377E-36 yard.

The following result demonstrates a very different issue, which will be returned to in the conclusion. It is human nature to expect a conversion such as 50 mile to metric to come out as a result in kilometres. However, due to the mathematical basis of the algorithm used, it does not!

Input	Output
50 mile to metric	0.0804672 megametre

F.2.3 Time-limited Upload

One requirement that has not been tested so far is the system's ability to retain user-uploaded CDs for a finite period of time, namely around 30 minutes. As specified in the design, this requirement is not technically met, but the design has it that files that have been uploaded but are older than 30 minutes should be deleted. This was tested by uploading units_beer1, and attempting to make use of it periodically over a 35 minute period. It was found that the system no longer recognised the units after approximately 30 minutes, although the front-end still offered them as suggestions until after the back-end had confirmed that it no longer knew about them, that is, one more conversion attempt.

F.2.4 Reverse Conversion link

Other features of the front-end also require testing, although it is not always possible to write a specific test for them. For example, a feature not specified in the requirements was the "Reverse this Conversion" link. This has been observed to work with various arbitrarily complicated units.

F.2.5 Suggestions

It was found that the system offers suggestions, and it was verified that these are loaded from the files created by the back-end of the system by deleting the files, which caused a warning to appear on the front-end of the system (and no suggestions), until the back-end had been run again, regenerating the files.

F.2.6 Web Content Guidelines

One of the requirements was to make the web page accessible and usable. The section shows the result of checking the page against the guidelines at World Wide Web Consortium (1999), although, as stated in the Literature Survey, Section 2.11, many of the points are not relevant to the system.

Guideline	**Pass/Fail**	**Notes**
1.1	-	No visual content used
1.2	-	No visual content used
1.3	-	No visual content used
1.4	-	No visual content used
1.5	-	No visual content used
2.1	Pass	Colour barely used
2.2	Pass	Colour barely used
3.1	Pass	No images used
3.2	Pass	Using W3C's validator
3.3	Pass	Used a little
3.4	-	Not used
3.5	Pass	H1 used
3.6	-	Not used
3.7	-	Not used
4.1	-	Only one natural language used
4.2	-	No abbreviations used
4.3	Pass	English specified
5.1	-	Not used
5.2	-	Not used
5.3	Pass	Table used for upload boxes, can be linearised
5.4	Pass	
5.5	Pass	
5.6	-	
6.1	Pass	
6.2	Fail	Dynamic content cannot be recreated statically without incredibly cluttered display, so it was decided the benefit gained from having the content was too small to be worthwhile
6.3	Fail	Files cannot be uploaded without JavaScript support, but conversions work.
6.4	Pass	onSelect used in addition to onClick
6.5	Fail	Page works without dynamic content, except for uploading files
7.1	-	Not used
7.2	-	Not used
7.3	Pass	Suggestions can be turned off, only update as user types when on
7.4	Pass	
7.5	Pass	
8.1	-	Not used
9.1	-	Not used
9.2	Pass	
9.3	Pass	
9.4	Pass	
9.5	-	No links need an access key
10.1	Pass	No pop-ups
10.2	Pass	
10.3	-	Not used
10.4	-	No longer relevant
10.5	Pass	
11.1	Pass	XHTML & CSS used
11.2	Pass	
11.3	Pass	
11.4	-	Not needed
12.1	-	Not used
12.2	-	Not used
12.3	-	Not needed
12.4	Pass	
13.1	Pass	
13.2	-	Not needed
13.3	-	Only a single page
13.4	Pass	
13.5	-	Not used
13.6	-	Not used
13.7	-	Not used
13.8	Pass	
13.9	-	Single page
13.10	-	Not used
14.1	Pass	
14.2	-	Not needed
14.3	-	Only one page

F.2.7 Summary of Requirements comparison for Front-end

Table F.4: Summary of whether front-end met its requirements

Requirement	Result	Notes
Functional Requirements		
1	N/A	The front-end has no concept of what is on the OpenMath website
1(a)	Pass	
1(a)i	Pass	The front-end takes the output from the back-end and outputs an appropriate message
1(a)i.A	Pass	File upload is offered
1(b)	N/A	
1(c)	N/A	This requirement cannot be tested, as there is no write access to the OpenMath website. However, it is believed that if the symbols get added to cdindex.html, they will be found.
2	Pass	The front-end uploads user-supplied OCD/STS files for use by the back-end
2(a)	Pass	The front-end uploads files supplied by the user, but makes no attempt to ensure names are unique
2(a)i	Pass	The user can upload files during a conversion, or independently of a conversion
2(a)ii	N/A	This was ruled out in an earlier stage
2(b)	N/A	
2(b)i	N/A	This was decided against, so does not occur
2(b)i.A	N/A	
2(b)ii	N/A	
2(c)	Pass	If the back-end returns an error, the front-end uses this to give a message explaining where in the file
2(c)i	Pass	
3	N/A	
3(a)	N/A	
3(b)	Pass	
5	Pass	A user-friendly message is returned
5(a)	Pass	The returned message explains the problem
5(a)i	Pass	Although it does not happen in the error state, when the user is typing, suggestions are offered
9	Pass	Suggestions are offered as the user types
10	Pass	A drop-down list of the integers from 1–15 inclusive is provided
Non-Functional Requirements		
5	Pass	The front-end does very little processing, so takes a negligible amount of time to process the result from the back-end
5(a)	Pass	Errors are processed and messages generated in this time
6	Pass	The time taken is not significantly longer than a conversion of built-in units
6(a)	Pass	
6(b)	Pass	
7	Pass	
7(a)	N/A	
7(b)	Pass	
9	Pass	Errors detail what the problem is, and what, if anything, the user could do to resolve it.
12	N/A	This has not been tested, but standards have been followed
Domain Requirements		
1	Pass	
1(a)	Pass	”label” tags are used to help screen readers, input-device independent access to features
1(a)i	Pass	
1(a)ii	Pass	
1(b)	N/A	

F.3 System Testing Results

Although it has already been seen that the front-end and back-end seem to work together, because the error messages came out appropriately, it is still necessary to ensure that some of the other features also perform correctly. For example, when the user types unit names into the boxes, the system should offer suggestions, providing the option has not been switched off. This has been observed to work, with units disappearing from the list as soon as they either do not start with what the user has typed, or they match it exactly. These appear as links, which the page explains how to navigate with the keyboard, or the mouse can be used as well. Selecting one of the links puts the unit selected as the last word in the text box. Selecting the checkbox to disable the suggestions does indeed cause the suggestions to be disabled, with a message appearing stating this.

F.3.1 Summary of Requirements comparison for System

In this section, we review how the system as a whole met its requirements, with notes detailing where it is not a simple consequence of either the front- or back-ends passing the requirement.

Table F.5: Summary of whether System met its requirements

Requirement	Result	Notes
Functional Requirements		
1	PASS	
1(a)	PASS	
1(a)i	PASS	
1(a)i.A	PASS	
1(b)	PASS	
1(c)	N/A	This requirement cannot be tested, as there is no write access to the OpenMath website. However, it is believed that if the symbols get added to cdindex.html, they will be found.
2	PASS	
2(a)	PASS	
2(a)i	PASS	
2(a)ii	N/A	
2(b)	PASS	
2(b)i	Fail	
2(b)i.A	N/A	
2(b)ii	N/A	
2(c)	PASS	
2(c)i	PASS	
3	N/A	
3(a)	N/A	
3(b)	PASS	
4	N/A	
4(a)	N/A	
5	PASS	The front-end uses the return value from the back-end to determine what happened, an outputs a result accordingly.
5(a)	PASS	The front-end explains the problem
5(a)i	PASS	Suggestions are offered at the time of input, thus hopefully reducing this type of user error.

6	PASS	
6(a)	PASS	
6(a)i	PASS	
7	N/A	Abbreviations have not been implemented
7(a)	N/A	
7(b)	N/A	
8	FAIL	Not in design
8(a)	FAIL	Not in design
9	PASS	The suggestions list generated by the back-end includes these units, and is used by the front-end
10	PASS	
Non-Functional Requirements		
5	PASS	
5(a)	PASS	
6	PASS	
6(a)	PASS	
6(b)	PASS	
7	PASS	
7(a)	N/A	
7(b)	PASS	
8	PASS	
8(a)	PASS	
9	PASS	Messages are generated by front- and back-end in combination
10	PASS	
11	PASS	
12	N/A	
Domain Requirements		
1	PASS	
1(a)	PASS	
1(a)i	PASS	
1(a)ii	PASS	
1(b)	N/A	

Appendix G

Project Proposal

Prior to embarking on this project, a proposal was written, and this is included here for completeness. It details, amongst other items, reasons for language choice.

Project Proposal

Jonathan Stratford

October 2007

Contents

1 Title

Creating an extensible Unit Converter using OpenMath as the representation of the semantics of the units.

2 Project Description

The idea behind this project is to create a program which, through the provision of unit definitions written in OpenMath, can convert between units. The use of OpenMath over another system (e.g. that used by Google, or convertit.com) gives several advantages. One is that the STS in OpenMath, which categorises units (Davenport 2000, Davenport & Naylor 2003) by their type (for example, whether it is a mass or a length) can be used to "sanity check" the input: if the user asks for "10 miles in kilogrammes", the program should report an error accordingly. Such a program could be infinitely extensible – a specialist user could add units that are not in general use, but are required for their purposes. Also, the program should be able to process e.g. "50lb into kilos" (both understanding lb as pounds and kilos as kilogrammes). In addition, the program should be able to understand such terms as "3 microns" as "3 micrometres", and also be able to convert "50 miles in metric" into a sensible result (e.g. kilometres). It would be useful for the user to be able to choose which units will be used (for an "into metric/imperial" conversion), and for example decide whether 0.5km should be represented as such, or as 500m. Ideally, the system should have a web front-end, so that it can (potentially) be accessed from anywhere.

Therefore a justification for this project is that a user can add units that they require themselves, they can choose the output units, and the program can tell if the units chosen are incompatible (e.g. length and mass).

2.1 Aims

The main aim is to produce a program which allows a user to convert between those units built into OpenMath, as well as any user-defined units.

2.2 Objectives

- Research working with units in OpenMath
- Research writing OpenMath Content Dictionaries
- Produce a program that can take user input for a given conversion in natural language, and produce the result. Both input and output (if metric) may make use of prefixes (e.g. kilogramme) and the program should interpret/output these correctly.

- Produce a program that allows the user to easily add new unit definitions in OpenMath, for subsequent use by the program.

3 Things to learn

Before getting started on this project, I will need to thoroughly research OpenMath, as I've not used it before. Additionally, I need to look into a complementary system, "MathML", as there are papers of relevance on that language. I may also need to learn a new programming language, or part of one, depending on which one I ultimately decide to use.

4 Requirements

From my early examination of the problem, I have arrived at the following set of requirements. Some are absolute requirements (indicated by "must"), while others are of less importance (marked by "should" and "may", in order of importance).

4.1 Functional Requirements

1. The system must correctly convert units built into OpenMath.
2. The system must allow for further unit definitions to be added, in OpenMath XML syntax.
3. The system must be able to understand a natural language expression in English for conversion (e.g. 5 miles in km).
4. The system must flag up an appropriate error if the user attempts to convert between two incompatible units.
5. The system must be able to convert to appropriate units in a given measurement standard (e.g. converting "miles into metric" would (probably) choose km as appropriate).
6. The system should correctly process common abbreviations/pseudonyms for any units that it "knows" about (e.g. kg, kilos for kilogrammes; microns, um for micrometres - or better still μm - although the user may be less likely to type the latter, and therefore support is a lower priority). This could be performed via one or more unit_abbreviations CDs, which I will need to write.
7. The system should have a web-based interface.
8. The system may offer the user the option of deciding whether, for example, 500m is represented as such, or as 0.5km.

4.2 Non-functional Requirements

1. The system must not take longer than six months to complete.
2. The system must only require one programmer
3. The system must be properly documented.
4. The project should result in several new CDs being submitted for approval, via Professor Davenport.

4.2.1 Programming Language(s)/System(s) use

This project does not inherently require a particular programming language. Going on the requirements previously listed, I think my choice should be influenced by the ease with which the language can parse XML (as the human-readable OpenMath format is an XML format (Buswell et al. 2004)) and the ease with which a web-based front-end can be included. Something that may become a consideration is whether I already know the language – as time will be a limiting factor on this project, I think it would be sensible to use a language I already know unless there's a good reason to use one that I would need to learn instead. Having thought about the options, I have decided I can either implement it as an integrated web service or as a web-based front-end which calls a separate command-line-based back-end, or a web-based set of pages that implement the entire functionality. With this in mind, I think my options currently are:

PHP PHP could be used either just as a front-end, or to provide the entire functionality.

Front-end only Using PHP for web-based front-end only, then writing a command line program in another language to perform all the processing, passing back to PHP to display the result. This has the advantage of easily separating the user interface from the functionality.

Whole functionality Using PHP to display the front-end, but also performing all the processing, before outputting the result. This could make the code more complicated, but it would be easier to only have to deal with a single language.

Web Service Using a web service for the back-end would mean that other programs could access the functionality, so that a wider variety of users could use my program. There are two available options that come under this category:

Using .NET Using the .NET framework, I could quite easily have a basic web service up and running, to which I would have to add my functionality. However, I know from experience that because a lot of the details are hidden, if something goes wrong it can be very difficult to track down the problem.

Using JSP I have no experience of using this (although I have knowledge of Java, which will help). Also, I do not think I have time to pursue this technology.

Back-end options If I choose to use a web scripting language such as PHP as the front-end, I could then write the back-end in any language I choose. The options for language choice, which is based on those languages that are suitable and I have access to, are as follows:

C for back-end This is the quickest-running language for the back-end, but can be fiddly doing string manipulation and memory allocation, and I would have to find an XML parser (there is no point writing my own).

Java for back-end Although this is a relatively slow-running language, it is easier to write than C. Also, has several XML parsers as standard.

C# for back-end This language runs almost as quickly as C, and is possibly easier to write than Java. It also has an XML parser built in that I am quite experienced at using.

Conclusion For these reasons, at this early stage, I am choosing to write the system in a combination of PHP for the front-end and C# for the back-end.

5 Resources

I will require a number of resources to complete this project; these are summarised below.

5.1 Hardware Resources

- A computer capable of using the Internet to acquire OpenMath Content Dictionary definitions, and running a webserver

5.2 Software Resources

- A Web server
- A compiler and editor for programming

- Typesetting program
- XML Parser in language I use
- (Possibly) A graphics editor

5.3 Personnel Resources

- Myself
- My Supervisor, Professor James Davenport

5.4 Literature Resources

- Papers on OpenMath
- Literature on units in general (Physics textbooks?)
- Papers on units in MathML
- Papers on problems involving unit conversion (or lack thereof)
- Resources on programming language of choice (e.g. XML parser)

5.5 Time Resources

- 30 weeks

6 Project Plan

This project contains a number of milestones and deliverables. These are summarised in table 1.

Event	Deadline	Project Week
Literature Survey handin	10/12/07	11
Progress Check handin	w.b. 04/02/07	19
Poster Presentation	w.b. 10/03/07	24
Final handin	28/04/07	31

Table 1: List of deliveries and important dates

With a view to completing the required work in time for these deliverables, I have created the Gantt Chart shown in Figure 1.

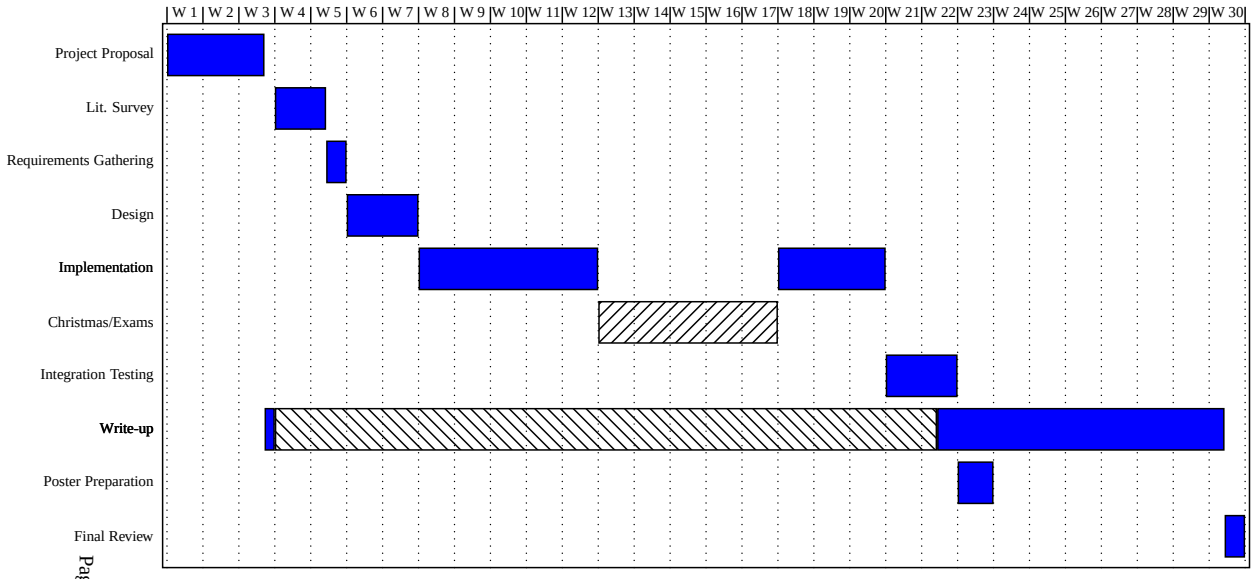


Figure 1: Preliminary Gantt Chart for project. Week 1 commencing 01/10/07, Week 30 commencing 21/04/08

References

Buswell, S., Caprotti, O., Carlisle, D. P., Dewar, M. C., Gatano, M. & Kohlhase, M. (2004), *OpenMath Standard 2.0*.
URL: *http://www.openmath.org/standard/om20-2004-06-30/omstd20.pdf*

Davenport, J. (2000), *A Small openMath Type System*.

Davenport, J. H. & Naylor, W. (2003), *Units and Dimensions in OpenMath*.

7 Supervisor agreement

I, Professor James Davenport, agree to supervise Jonathan Stratford on the Final Year project described in this document.

Signed _____ Date _____

Appendix H

Unit Knowledge Management

As a result of work in this dissertation, we decided to write a separate paper, which we submitted to MKM2008. This paper is included here.

Unit Knowledge Management

Jonathan Stratford & James H. Davenport

Department of Computer Science
University of Bath, Bath BA2 7AY, United Kingdom
{jds21,J.H.Davenport}@bath.ac.uk

Abstract. In [9], various observations on the handling of (physical) units in OpenMath were made. In this paper, we update those observations, and make some comments based on a working unit converter [20] that, because of its OpenMath-based design, is modular, extensible and reflective.

1 Introduction

For the purposes of this paper, we define a **unit** of measurement as *any determinate quantity, dimension, or magnitude adopted as a basis or standard of measurement for other quantities of the same kind and in terms of which their magnitude is calculated or expressed* [18, unit].

There have been many famous examples where unit conversion was not undertaken, or where it was incorrectly calculated. The Gimli Glider [17, 22], as it became known, was a (then) new Boeing 767 plane, which, during what should have been a routine flight in 1983, ran out of fuel just over halfway to its intended destination. The ensuing investigation established that an incorrect conversion had been performed, leading to a woefully insufficient fuel payload, because the aircraft was one of the first of its kind to use a metric measure of fuel, and the refuellers had used an imperial conversion instead of the correct metric one. In addition, although a second check was carried out between legs of the flight, the same incorrect conversion was used.

Large organisations such as NASA are not immune to such problems [16]. Software controlling the thrusters on the Mars Climate Orbiter was configured to use imperial units, while ground control, and the other parts of the space craft, interpreted values as if they were metric. This led to the orbiter entering an incorrect orbit too close to Mars, and ultimately to its being destroyed.

Again, even widespread systems such as Google can get this wrong — see the example in section 4 — as can attempts such as OntoWeb to "understand" MathML in terms of simple structures such as RDF [6] (section 4.3).

2 Prior Work on Semantics of Units

2.1 OpenMath

OpenMath [3] is a standard for representing mathematical semantics. It differs from the existing versions[1] of Content MathML [4, 5] in being *extensible*: new Content Dictionaries (CDs) can add new OpenMath symbols, known as OMS, and can prescribe their semantic, via Formal Mathematical Poperties (FMPs). In contrast, OpenMath *variables*, known as OMV, are purely names.

OpenMath is essentially agnostic with respect to type systems. However, one particular one, the Simple Type System [8] is used to provide arity and similar information that is mechanical, and also information that is human-readable, but not currently machine processable, such as stating that `<OMS name="plus" cd ="arith1"/>` takes its arguments from, and returns an answer in, the *same* Abelian semigroup, by having the following signature.

```
<OMA>
 <OMS name="mapsto" cd="sts"/>
 <OMA>
  <OMS name="nassoc" cd="sts"/>
  <OMV name="AbelianSemiGroup"/>
 </OMA>
 <OMV name="AbelianSemiGroup"/>
</OMA>
```

2.2 Prior Work on Units in OpenMath

The major previous work on the semantics of units on OpenMath is [9]. This proposes several Content Dictionaries of units: `units_metric1`, `units_imperial1` and `units_us1`. These contain definitions of many common units covering a variety of dimensions (the dimensions themselves are defined in the Content Dictionary `dimensions1`) — metric (SI)[2] units are contained in `units_metric1`, for example. [9] suggests using the "usual" times operator (that stored in `arith1`) to represent a number in a particular unit — i.e. storing the value as the number

[1] Versions 1 and, to a lesser extent, 2. It is intended that OpenMath 3 and Content MathML 3 will have converged on this important point

[2] The system in [9] actually differs in one respect from the SI system in [12, 14]. [9] takes the fundamental unit of mass to be the gram, rather than the kilogram. This is necessary, as a slavish following of the general principles of [12] would lead to such absurdities as the millikilogram (see section 3.1 of this paper) rather than the gram. [12, section 3.2] explains the special rules for multiples of the kilogram, as follows.

> Names and symbols for decimal multiples and submultiples of the unit of mass are formed by attaching prefix names to the unit name "gram", and prefix symbols to the unit symbol "g" (CIPM 1967, Recommendation 2; PV, 35, 29 and *Metrologia*, 1968, 4, 45).

multiplied by the unit, with the unit following the value to which it refers. The suggestions for unit "implementation" in OpenMath are stated as being based on those used by a complementary mathematics display language, MathML — although not blindly; where the authors believe MathML has some deficiencies, these have been corrected. This document also specifies a reasonable way of connecting a prefix to a unit (described in section 3.1), thus defining `kilo` as a separate concept, which can then be used to construct `kilogram`.

2.3 Unit Converters

There are a great many unit converters publicly available online. These have a range of units and features. However, in all cases, they are monolithic, in that new units cannot be added to them by the user. In some senses, this means that they go against modularity and incrementality, and are not reflective, in that they do not know that other units exist. The following table summarises the availability of typical features in a cross-section of converters. For further details, see [20, chapter 2].

We consider the following unit converters.

1. `http://digitaldutch.com/unitconverter/`
2. `http://www.onlineconversion.com/`
3. `http://www.unitconversion.org/`
4. `http://www.convert-me.com/en/`
5. `http://online.unitconverterpro.com/`
6. `http://www.knovel.com/knovel2/unitconverter.jsp`
7. `http://www.chemie.fu-berlin.de/chemistry/general/units\_en.html`
8. `http://www.he.net/\textasciitildeseidel/Converter/`
9. `http://www.engnetglobal.com/tips/convert.asp`
10. `http://www.megaconverter.com/Mega2/`
11. `http://www.unitsconverter.net/`
12. `http://www.convertit.com/Go/ConvertIt/Measurement/`
13. `http://www.convertunits.com/`
14. Google calculator—`http://www.google.co.uk`

	Feature															
	Neg.	Inst.	Multi.	Cat.	Rev.	NL	Abb.	Comp.	Sug.	Prec.	Mod.	Copy	Calc.	B.E.	P-u	Comb.
1	✔	✔	✗	12	✔	✗	N/A	✔	✗	1–15dp	✔	✗	✗	✗	✗	✗
2	✔	✔	✗	27	✗	✔	✔	✔	✗	10dp	✔	✗	✔	✗	✗	✗
3	✔	✔	✔	79	N/A	✗	N/A	✔	✗	0–15dp	✔	✗	✔	✗	✗	✗
4	✔	✗	✔	22	N/A	✗	N/A	✔	✗	1–7sf	✔	✗	✗	✗	✗	✗
5	✔	✔	✗	76	✗	✗	N/A	✔	✗	11sf	✔	✗	✔	✗	✗	✗
6	✔	✗	✗	87	✗	✗	N/A	✔	✗	1-16sf	✔	✔	✗	✔	✗	✗
7	✗	✗	✗	✗	✔	✗	N/A	✔	✗	7sf	✗	✗	✗	✔	✗	✗
8	✔	✗	✔	✗	N/A	✗	N/A	✔	✗	15sf	✗	✗	✗	✔	✗	✗
9	✔	✔	✔	21	N/A	✔	N/A	✔	✔	7dp	✔	✗	✗	✗	✗	✗
10	✔	✔	✗	45	✔	✗	N/A	✔	✗	5dp	✔	✗	✗	✗	✔	✗
11	✔	✗	✔	10	N/A	✗	N/A	✔	✗	10dp	✔	✗	✗	✗	✗	✔
12	✔	✗	✗	✗	✗	✔	✔	✗	✔	15sf	✔	✗	✗	✔	✗	✗
13	✔	✔	✗	✗	✔	✔	✔	✔	✔	16sf	✔	✗	✔	✗	✗	✔
14	✔	✗	✗	✗	✗	✔	✔	✔	✗	9sf	✔	✗	✔	✔	✗	✔

Table 1. Summary of unit converter features.
Key: Neg.—Negative, Inst.—Instantaneous, Multi.—Multiple-at-once, Cat.—Categories, Rev.—Reverse conversion possible?, NL—Natural Language interpreter, Abb.—If NL, does it support abbreviations?, Comp.—Conversion of compatible units only, Sug.—Suggestions provided (based on user input)?, Prec.—Precision (range or specific), dp—decimal places, sf—significant figures, Mod.—Query can be easily modified after conversion, Copy—Copy-to-clipboard functionality provided, Calc.—can process calculations, B.E.—appears to call back-end program, P-u—option to open conversion in a pop-up window for use with other sites, Comb.—can process combinations of units at once. N/A—not applicable for this converter, for example the "reverse" of a conversion where Multi. is true is undefined.

3 Abbreviations and Prefixes

Units, particularly in the metric system, have a variety of abbreviations and prefixes. It is possible, as apparent in [13, section 5.3.5], to regard prefixed units as units in their own right, and introduce a unit `centimetre` with a formal property relating it to the `metre`, but this way lies, if not actual madness, vast repetition and the scope for error or inconsistency (who would remember to define the `yottapascal`?).

3.1 Prefixes

OpenMath therefore defines prefixes in the `units_siprefix1` CD, with FMPs to define the semantics, e.g. the following one for `peta`.

```
<OMA>
  <OMS name="eq" cd="relation1"/>
  <OMA>
    <OMS name="times" cd="arith1"/>
    <OMI> 1 </OMI>
    <OMA>
      <OMS name="prefix" cd="units_ops1"/>
      <OMS name="peta" cd="units_siprefix1"/>
      <OMV name="unit"/>
    </OMA>
  </OMA>
  <OMA>
    <OMS name="times" cd="arith1"/>
    <OMA>
      <OMS name="power" cd="arith1"/>
      <OMI> 10 </OMI>
      <OMI> 15 </OMI>
    </OMA>
    <OMV name="unit"/>
  </OMA>
</OMA>
</OMOBJ>
```

OpenMath uses a `prefix` operation (described as option 4 of [9, section 4]) to apply prefixes to OpenMath units. Its signature is given as follows.

```
<Signature name="prefix" >
<OMOBJ xmlns="http://www.openmath.org/OpenMath">
  <OMA>
    <OMS name="mapsto" cd="sts"/>
    <OMS cd="units_sts" name="prefix"/>
    <OMV name="dimension"/>
    <OMV name="dimension"/>
```

```
  </OMA>
</OMOBJ>
</Signature>
```

which can be seen as

$$\text{prefix} \times \text{unit} \to \text{unit}. \tag{1}$$

This has the slightly unfortunate property that it would allow, for example, 'millimicrometre', which is explicitly forbidden by [12, p. 122]. This could be solved by making the signature

$$\text{prefix} \times \text{unit} \to \text{prefixed unit}, \tag{2}$$

which should probably be done.

This construction also allows the use of prefixes with non-SI units, but this is in fact legitimate [12, p. 122].

3.2 Abbreviations

One issue not covered in [9] is that of abbreviations. Here we must confess to not having a completely worked-out and sensible solution yet. The following possibilities have been considered.

Alternative Definition in the same CD This would mean that, for example, as well as `units_metric1` having the symbol `metre`, it would also have `m`. These would be linked via a FMP saying that the two were equal. Similarly, we would have prefixes `k` as well as `kilo`.

Pro A small extension of [9].
Con Allows "mixed" units such as `kilom` or `kmetre`, which are (implicitly) forbidden in [12].
Con No built-in way of knowing which is the full name and which is the abbreviation.

Alternative Definition in different CDs This would mean that `units_metric1` would have the symbol `metre`, and a new CD, say `units_metricabbrev1`, would have the symbol `m`. Again, these would be linked via a FMP saying that the two were equal. We would also have a new CD, say `units_sipefixabbrev1`, containing the abbreviations for the prefixes, and a *different* operation for combining the two, say

```
<Signature name="prefixabbrev" >
<OMOBJ xmlns="http://www.openmath.org/OpenMath">
  <OMA>
    <OMS name="mapsto" cd="sts"/>
    <OMS cd="units_sts" name="prefixabbrev"/>
    <OMV name="dimensionabbrev"/>
```

```
    <OMV name="dimensionabbrev"/>
  </OMA>
</OMOBJ>
</Signature>
```

Pro Prevents 'hybrid' units.

Pro A converter such as [20] could output either full names or abbreviations ('symbols' in [12]) depending on which CDs were available on the output side.

Con Knowledge of which is the name and which is the abbreviation is still implicit — merely moved from the name of the symbol to the name of the CD. The linkage between the name of the CD and the fact that the symbol should be regarded as `<OMV name="dimensionabbrev"/>` would be outside the formal OpenMath system.

"This isn't an OpenMath problem" It could be argued that abbreviating units and prefixes isn't an OpenMath problem at all, but a presentation one. This is superficially tempting, but poses the question "Whose problem is it?" Do we need a new layer of software to deal with it? One interesting sub-question here is whether an ontology language such as OWL [7] would be better suited to expressing such concepts.

3.3 Non-SI Units

The reader will have noticed that the CD is called `units_metric1` rather than `units_si1`. This is deliberate, as it includes the `litre`, which is explicitly *not* an SI unit [12, Table 6, note (f)]. What of the other units in [12, Tables 6, 8]?

tonne (alias 'metric ton') [12, Table 6] This is essentially an alias for the megagram, and as such does not take prefixes[3]. If the "different CDs" approach above were to be adopted, this could be in yet another CD, say `units_metricmisc1`, on which no prefixing operated.

hectare As $10^4 m^2$, this is in a very similar category to the tonne, and again does not take prefixes. The only question might be whether we ought to start with the `are` instead, but it is possible to argue that the `are` is obsolete, and conveys no advantage over the square decameter. If `litre_pre1964` moves to a different CD, we could reasonably leave the `are` there as well.

bar Similarly.

ångström Similarly.

nautical mile (= 1852m) Similarly.

knot ($=\frac{1852}{3600}$ m/s) Similarly. This is also an excellent argument for the representaton of definitional conversions (section 5.1) as exact fractions.

[3] The reader may ask "what about the megaton(ne)?" This is, of course, the mega'ton of TNT equivalent', and is not a unit of mass at all, but rather of energy, and is in fact 4.184 petajoules [2, Appendix B.8], where the figure 4.184 is definitional in the sense of section 5.1.

4 Not All Dimensions are Monoids

[9] assumed, implicitly, that all physical dimensions could be regarded as (Abelian) monoids, in the sense that they could be added, and hence multiplied by integers. This is in fact not the case.

4.1 The Temperature Problem

One problem was not addressed in [9], but has been observed elsewhere [1, `Celsius#_note-10`], viz. that temperatures are not the same thing as temperature intervals. This confusion is widespread, as evidenced by the Google calculator's ability to produce computational absurdities such as

```
(1 degree Celsius) plus (1 degree Celsius) = 275.15 degrees Celsius
```

Possibly the best explanation of the difference is in [2, Appendix B.9][4].

4.2 To Monoid or not to Monoid

We can ask whether this is a peculiarity of temperature. The answer is in fact that it is not.

Most units form (Abelian) monoids, i.e. they can be added: 2 tonnes + 3 tonnes = 5 tonnes etc. *Non-relative*[5] temperatures are one obvious counter-example: $2°\mathrm{F} + 3°\mathrm{F} \neq 5°\mathrm{F}$ or indeed any other temperature. The point is that *relative* temperatures, as in "A is ten degrees hotter than B", *are* additive, in the sense that if "B is twenty degrees hotter than C", then indeed "A is thirty degrees hotter than C", *are* additive, as are non-relative plus relative, but two absolute temperatures are *not* additive.

The same problem manifests itself with other scales such as decibels. Strictly speaking, these are purely relative, but in practice are also used in an absolute way, as in "the sound level exceeded 85dB". Again, the relative units form a monoid, but the absolute units do not.

This forces us to rethink the concept of "dimension". Though not using the word here (it is used in section 1.3), these are defined in [12, section 1.2] as follows.

> The base quantities used in the SI are length, mass, time, electric current, thermodynamic temperature, amount of substance, and luminous intensity.

This implies that all masses, for example, have the same dimension, and can be treated algebraically in the same way. But, as we have seen, not all temperatures are the same, and indeed have different algebraic properties. Two relative

[4] `http://physics.nist.gov/Pubs/SP811/appenB9.html\#TEMPERATUREinterval`.

[5] Referring to 'absolute' temperatures would be likely to cause confusion, though that is what we mean in sense 10 of [18, absolute].

temperatures can be added, as in the example of A, B and C above. A relative temperature can be added to an absolute temperature, as in the following examples.

$$X \text{ was heated by } 10°\text{C}, \text{ from } 20°\text{C to } 30°\text{C}. \tag{3}$$

$$X \text{ was heated by } 10\text{K}, \text{ from } 20°\text{C to } 30°\text{C}. \tag{4}$$

$$X \text{ was heated by } 10°\text{C}, \text{ from } 293.15\text{K to } 303.15\text{K}. \tag{5}$$

$$X \text{ was heated by } 10\text{K}, \text{ from } 293.15\text{K to } 303.15\text{K}. \tag{6}$$

Equations (3) and (4) mean precisely the same thing[6], and this is obvious because, in the Content Dictionary defining relative temperatures, we state that the two are equal[7].

```
<OMA>
  <OMS name="eq" cd="relation1"/>
  <OMA>
    <OMS name="times" cd="arith1"/>
    <OMI> 1 </OMI>
    <OMS name="relative_Kelvin" cd="units_metric1"/>
  </OMA>
  <OMA>
    <OMS name="times" cd="arith1"/>
    <OMI> 1 </OMI>
    <OMS name="relative_Celsius" cd="units_metric1"/>
  </OMA>
</OMA>
```

Equations (3) and (5) also mean precisely the same thing, but this time we need to rely on the definitions of non-relative temperatures, as in the following[8],

```
<OMA>
  <OMS name="eq" cd="relation1"/>
  <OMA>
    <OMS name="times" cd="arith1"/>
    <OMI> 1 </OMI>
    <OMS name="degree_Kelvin" cd="units_metric1"/>
  </OMA>
```

[6] Similarly for equations (5) and (6).

[7] This is the standard OpenMath way of doing so for units. It might make more sense simply to declare that the two symbols were precisely equal — see the discussion in section 7.2. It could be argued that, since the two symbols are equal, we do not actually need to have both — a minimalist view. This is similar to the discussion about `<OMS name="Landauin" cd="asymp1"/>` in [11], and our conclusion would be the same — convenience of rendering outweights minimality.

[8] This differs from the currently-published `experimental` CD `units_metric1` in following the recommendation in section 5.1 that 273.15, as a defined number, should be represented as an element of $\mathbf{Q}$.

```
<OMA>
  <OMS name="minus" cd="arith1"/>
  <OMA>
    <OMS name="times" cd="arith1"/>
    <OMI> 1 </OMI>
    <OMS name="degree_Celsius" cd="units_metric1"/>
  </OMA>
  <OMA>
    <OMS name="divide" cd="arith1"/>
    <OMI>  27315 </OMI>
    <OMI>    100 </OMI>
  </OMA>
</OMA>
</OMA>
```

and need to do some actual arithmetic, as in [20].

The system of dimensions in [9] had, as the Simple Type System [8] signature of dimensions, the OpenMath `<OMV name="PhysicalDimension"/>`. As its format implies, this is a mere name with no formal semantic connotation. We therefore suggest that this be replaced by two objects: `MonoidDimension` for those cases where the dimension *does* represent an (additive) monoid, and `NonMonoidDimension`. While these *could* be OMVs as before, we believe that it would make more sense for them to be OMSs.

4.3 The Confusion is Widespread

It should be noted that the confusion between temperatures and relative temperatures manifests itself elsewhere in "web semantics". Consider an excerpt[9] from ontoworld's "approach to rewrite Content MathML so that it is expressable as RDF."

```
<owl:Class rdf:about="&phml;Temperature">
  <rdfs:subClassOf rdf:resource="&phml;PhysicalDimension"/>
</owl:Class>

<owl:Class rdf:about="&phml;TemperatureDifference">
  <rdfs:subClassOf rdf:resource="&phml;Temperature"/>
</owl:Class>
```

This could be argued to illustrate the difficulties of using a general-purpose language such as RDF beyond the semantics it is capable of handling, or, more simply perhaps, as an illustration of the fact that, since the *presentation* of temperatures and temperature intervals are the same, it is hard to distinguish the *semantics*, different though these may be.

[9] `http://ontoworld.org/wiki/Semantic_MathML/0.1\#Physical_Dimensions`.

5 Precision

5.1 Accuracy in the OpenMath

Conversion factors between units can be divided broadly into three categories.

Architected There are those that, at least conceptually, arose when the unit(s) were defined. All the metric prefixes fall into this category, as do conversions such as "3 feet = 1 yard", or even "1 rod = $5\frac{1}{2}$ yards". These conversions, and their inverses, clearly ought to be stored as elements of $\mathbf{Q}$, i.e. as OpenMath integers `OMI` or fractions thereof.

Experimental These are those that are truly determined by an experiment, such as the measurement of a standard of length in one system in terms of another such. An obvious example is those units that involve g, as in "1 slug $\approx$ 32.17405 pounds". These are probably best represented by means of floating-point numbers, `OMF`.

The reader might object that, since these items are only approximate, we should represent them by intervals, which are well-handled by OpenMath, as in the CD `interval1`. This is a plausible point. We happen to disagree with it, for the reasons about to be given, but nevertheless it is fair to say that more usage of these factors is called for before a definitive decision can be made.

- Manipulation of intervals is not conservative unless it is done symbolically — [10]. Hence, if g were to be represented by an interval, say $[32, 33]$ (absurdly wide, but this makes the point better), one slug would be $[32, 33]$ pounds, which, on conversion back, would become $[\frac{32}{33} \approx 0.97, \frac{33}{32} \approx 1.03]$ slugs.
- Definitions in OpenMath are intended to be permanent, so an increase in precision would have to lead to a change in the formal definition.
- Experimentalists tend not to work in terms of intervals, but in terms of the standard accuracy [15]. It would be a fair argument, though, to say that there *ought to be* OpenMath interval types capable of representing these.

Definitional These are those that started life as experimental, but have since been adopted as architected definitions. An obvious example is "1 yard = 0.9144 metre", which was adopted as a formal definition, replacing the previous experimental result[10] of "1 metre $\approx$ 39.370147 inches" [19] in 1959.

Another example would be the value of "absolute zero", in the days of an independent celsius scale, which was about -273.15°C. Nowadays, this is fixed as precsiely this value, or, more accurately, the concept of °C is defined in terms of absolute zero and the number 273.15 [12, 2.1.1.5].

[10] It is worth noting that [19] describes this as "1 yard = 0.91439841 metres", [1, Imperial_units] as 0.914398416 metres, and an accurate conversion of the headline figure in [19] is .9143984146. This illustrates the general point that a number and its reciprocal are unlikely both to be exact decimals.

We now believe that all *definitional* numbers occurring in unit conversions, as well as those architected, should be expressed as elements of **Q**, i.e. as (fractions of) OMI. Hence the 0.9144 mentioned above should in fact be encoded as

```
<OMA>
  <OMS name="divide" cd="arith1"/>
  <OMI>  9144 </OMI>
  <OMI> 10000 </OMI>
</OMA>
```

This suggestion is well-characterised by the foot. Thus U.S. survey foot is defined[11] as $\frac{1200}{3937} \approx .3048006096$m, whereas the 'international' foot is defined as *precisely* 0.3048m. The difference is only just detectable in IEEE (single-precision) floating-point, and is best stored exactly.

5.2 Precision of Display

There is also the issue of how much precision to display in the result. In general terms, the result should not be more precise than the least precise value used in the calculation. [20] currently supplies the entire result from the calculation, with a user-controllable "significant figures" level as part of the system's front-end. This was chosen on the basis that several alternatives considered appeared nonsensical or unreasonably difficult to implement or make firm decisions about. For example, internally, fractions (which in OpenMath are comprised of two infinite precision integers) are stored as finite-precision floating point numbers. It is impossible to tell, when presented with such a floating point number, whether it was made as such (again, OpenMath Floats are of infinite precision) or whether it came from a fraction; in both these cases rounding would not be required, or if it was the result of a calculation, in which case rounding would be necessary. A non-sensical answer would clearly result if values were rounded during the calculation, and due to the aforementioned unknowable fact of where the value came from, it would also be impossible to maintain an internal counter of to how many significant figures the end result would be reasonable. With the chosen approach, a currently unanswered question regards the number of significant figures to display in a result such as `10 metre is 1 rod 5 yard 1 foot 3 inch 0.700787401574787 mil`. Should the number of significant figures only cover the last part of the result?

6 The two meanings of "obsolete"

According to the OpenMath standard [3], a content dictionary can be declared to be `obsolete`. This facility is needed so that, when an area of OpenMath gets rewritten in a (hopefully) better way, the semantics of existing OpenMath objects are preserved. However, there has been no need to deploy it yet. It is

[11] U.S. Metric Law of 1866.

a feature of OpenMath[12] that this takes place at the content dictionary level, rather than the symbol level.

However, when we say that

```
<OMS name="litre_pre1964" cd="units_metric1"/>
```

is "obsolete", we do *not* mean that it is obsolete as an OpenMath symbol, rather that it is a current OpenMath symbol denoting an obsolete unit of measurement, and therefore that it it should be in an `official` CD. Does this matter? There are two views.

No This is the view of [9]. It is a unit, which may still be encountered as old texts/experiments etc. are analysed, so should be present.

Yes In [20] we produced a unit converter that attempted to produce the "best" fit to a given input. Hence, as 176.0563390 `pint`s converts to 100.0000006 `litre`s, but also 99.99720068 `litre_pre1964`s, the latter conversion would, much to the user's surprise (and indeed ours on first encountering this issue), be preferred.
Similarly, as 10 `metre`s is 10.93613298 `yard`s, but 1.988387814 `rod`s, the latter will again be preferred[13].

From the point of view of 'user-friendliness', we are inclined to sympathise with [20], and state that obsolete units belong in separate CDs, in particular that `litre_pre1964` should be moved from `units_metric1` to, say, `units_metricobs` before `units_metric1` becomes `official`.

7 Conclusion

We conclude that it is possible to use the OpenMath unit system (or ontology, as one might call it) of [9] to produce a serious and, unlike others, *extensible* unit converter, as in [20].

7.1 Recommendations for OpenMath Unit/Dimension CDs

The most important recommendation is a recognition that some (in our sense of the word) dimensions are (additive) monoids, and some are not, as outlined in section 4.

1. Move `litre_pre1964` into a different CD, which is an `official` CD of "obsolete" units. Similar steps should be taken for "obsolete" imperial units.
2. Fix `dimensions1` so as to have a definition for power.
3. Delete `metre_squared` from the `units_metric1`. It is anomalous (why isn't there `metre_cubed`, and why doesn't `units_imperial1` have `foot_squared`?) and tempts a piece of software (such as earlier versions of [20]) into creating units such as

[12] At least at version 2. This may change in version 3.

[13] which will in fact come out as `10 metre is 1 rod 5 yard 1 foot 3 inch 0.700787401574787 mil`.

```
<OMA>
  <OMS name="prefix" cd="units_ops1"/>
  <OMS name="deci" cd="units_siprefix1"/>
  <OMs name="metre_squared" cd="units_metric1"/>
</OMA>
```

which is a deci(metre^2), as opposed to a (decimetre)^2, and is illegal [12, p. 121].

4. `units_imperial1` is missing units such as `inch`, which need to be added.
5. Add[1] a CD for U.S. units, where different (e.g. for volume). Move U.S. Survey units[14], currently in `units_imperial1` into this.
6. Add a CD for E.U. units, where different. The only case known to the authors is the `therm`, which comes in both U.S. and E.U. variants. [2, footnote 25] states the following.

   > Although the therm (EC), which is based on the International Table Btu, is frequently used by engineers in the United States, the therm (U.S.) is the legal unit used by the U.S. natural gas industry.

   The difference is about 0.02%.
7. Update all the semantics in the world of OpenMath units so as to adhere to the principles of section 5.1, in particular definitional numbers should be expressed as elements of $\mathbf{Q}$, i.e. as (fractions of) OMI.
8. Sort out electrical energy definitions and other suggestions in [9].
9. Modify the signature of `prefix`, as described in section 3.1, from (1) to (2).
10. Update the definition of `pascal` to include an FMP: currently missing.

EdNote(1)

7.2 Further Considerations

We saw, in section 4.3, that the sort of semantics of RDF [6] are inadequate to convey the relationship between, for example, relative temperature and non-relative temperature. However, the OpenMath required to state that

```
<OMS name="relative_Kelvin" cd="units_metric1"/>
```

and

```
<OMS name="relative_Celsius" cd="units_metric1"/>
```

mean *precisely* the same thing is clumsy, and requires OpenMath-capable reasoning whereas all that is needed *in this case* is RDF-like, or OWL-like, reasoning.

We can also ask whether OWL would not be better at solving the abbreviations problem than OpenMath (see section 3.2).

Some units (`calendar_year` is the notable example) have multiple FMPs, whereas most of the other secondary units have only one, which is essentially a *defining* mathematical property in the sense [21, I, p. 11] that the definiens can be completely replaced by the definiendum. Making the distinction clear, as has been proposed elsewhere in the OpenMath community, would be a step forward.

[1] EdNote: JDS points out there already is one, and U.S. Survey units appear to be confused.

[14] See the discussion at the end of section 5.1 for the (small) difference.

7.3 Unsolved Problems

We see two currently unsolved problems.

1. The abbreviations issue — section 3.2.
2. The difference between `length` and `displacement` as dimensions in `dimensions1`.

References

1. Anonymous. Wikipedia, English. `http://www.wikipedia.org`, 2008.
2. B. N. Taylor. `http://physics.nist.gov/Pubs/SP811`, may 2007.
3. S. Buswell, O. Caprotti, D. P. Carlisle, M. C. Dewar, M. Gaëtano, and M. Kohlhase. OpenMath Standard 2.0. `http://www.openmath.org/standard/om20-2004-06-30/omstd20.pdf`, 2004.
4. World-Wide Web Consortium. Mathematical Markup Language (MathML[tm]) 1.01 Specification. `http://www.w3.org/TR/MathML/`, 1999.
5. World-Wide Web Consortium. Mathematical Markup Language (MathML) Version 2.0 (Second Edition). `http://www.w3.org/TR/MathML2/`, 2003.
6. World-Wide Web Consortium. RDF Semantics. `http://www.w3.org/TR/2004/REC-rdf-mt-20040210/`, 2004.
7. World-Wide Web Consortium. Web Ontology Language OWL. `http://www.w3.org/2004/OWL/`, 2004.
8. J. H. Davenport. A Small OpenMath Type System. *ACM SIGSAM Bulletin*, 34(2):16–21, 2000.
9. J. H. Davenport and W.A. Naylor. Units and Dimensions in OpenMath. `http://www.openmath.org/documents/Units.pdf`, 2003.
10. J.H. Davenport and H.-C. Fischer. Manipulation of Expressions. *Improving Floating-Point Programming*, pages 149–167, 1990.
11. J.H. Davenport and P. Libbrecht. The Freedom to Extend OpenMath and its Utility. *To appear in Mathematics in Computer Science*, 2008.
12. Organisation Intergouvernementale de la Convention du Mètre. The International System of Units (SI) 8th edition. `http://www.bipm.org/utils/common/pdf/si_brochure_8_en.pdf`.
13. World-Wide Web Consortium (ed. D.W. Harder and S. Devitt). Units in MathML. `http://www.w3.org/Math/Documents/Notes/units.xml`, 2003.
14. IEEE/ASTM. SI 10-1997 Standard for the Use of the International System of Units (SI): The Modern Metric System. *IEEE Inc.*, 1997.
15. International Standards Organisation. *Guide to the Expression of Uncertainty in Measurement*, 1995.
16. Mars Climate Orbiter Mishap Investigation Board. Mars climate orbiter: Phase i report. `ftp://ftp.hq.nasa.gov/pub/pao/reports/1999/MCO\_report.pdf`, November 1999.
17. Wade H. Nelson. The Gimli Glider. *Soaring Magazine*, 1997.
18. Oxford University Press. Oxford English Dictionary. `http://dictionary.oed.com/entrance.dtl`, 2008.
19. J. E. Sears, W. H. Johnson, and H. L. P. Jolly. A New Determination of the Ratio of the Imperial Standard Yard to the International Prototype Metre. *Phil. Trans. Roy. Soc. A*, 227:281–315, 1928.
20. Jonathan Stratford. An OpenMath-based Units Converter. Dissertation for B.Sc. in Computer Science, University of Bath, 2008.

21. A.N. Whitehead and B. Russell. *Principia Mathematica*. Cambridge University Press, 1910.
22. Merran Williams. The 156-tonne Gimli Glider. *Flight Safety Australia*, 2003.

Appendix I

User Documentation

I.1 Introduction

This chapter will detail how to set up the system to be used, or so that work on it can be continued. Due to the set-up used, the front-end had to be run from a different drive to the back-end, and this unfortunately meant that absolute paths were needed in the front-end to locate the back-end. These would need changing when trying to run the system, but this is easily rectified, and can be converted to a relative path if the files are on the same drive.

I.2 Back-end

Copy the complete OpenMathConverter directory tree from the CD. If just running the program, only the OpenMathConverter/bin/Release directory is needed. However, it should be borne in mind when choosing where to place the directory, if just taking the Release directory, that the program looks for an upload directory 3 levels up from itself in the directory tree—this is so that the directory was in the top level of the solution. In addition, it writes the dataFiles directory, containing the files it downloads, to this location.

To open the solution, open the OpenMathConverter.sln file.

I.2.1 Documentation

Besides the contents of this document, the program is set up to generate XML documentation, when compiled. This file contains details of all the methods in all the classes that have associated documentation comments (starting with ///).

I.3 Front-end

This requires slightly more setup. After copying the units directory to the directory used by the web-server, the code needs to be modified so that it can find the back-end executable.